TESIS – IF 185401

NON-PLAYABLE CHARACTER ADAPTIF PADA *GAME* RPG MENGGUNAKAN METODE *LOGARITHMIC LEARNING FOR GENERALIZED CLASSIFIER NEURAL NETWORK* (L-GCNN)

IZZA MABRUROH
NRP. 5116201012

DOSEN PEMBIMBING

Dr. Eng. Darlis Herumurti, S.Kom.,M.Kom.
NIP. 197712172003121001

PROGRAM MAGISTER
BIDANG KEAHLIAN INTERAKSI, GRAFIS DAN SENI
DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2019

(Halaman sengaja dikosongkan)

TESIS – IF 185401

ADAPTIVE NON-PLAYABLE CHARACTER FOR RPG GAME USING LOGARITHMIC LEARNING FOR GENERALIZED CLASSIFIER NEURAL NETWORK (L-GCNN) METHOD

IZZA MABRUROH
NRP. 5116201012

SUPERVISOR

Dr. Eng. Darlis Herumurti, S.Kom.,M.Kom.
NIP. 197712172003121001

MASTER PROGRAM
THE EXPERT OF GRAPHICHS AND ART INTERACTION
DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2019

(Halaman sengaja dikosongkan)

LEMBAR PENGESAHAN

Tesis ini disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M. Kom)
di

Institut Teknologi Sepuluh Nopember Surabaya

oleh:
IZZA MABRUROH
Nrp. 5116201012

Dengan judul:
Non-Playable Character Adaptif Pada *Game* RPG Menggunakan Metode *Logarithmic Learning For Generalized Classifier Neural Network* (L-GCNN)

Tanggal Ujian : 18 Januari 2019
Periode Wisuda : 2019 Ganjil

Disetujui oleh:

Dr.Eng. Darlis Herumurti, S.Kom, M.Kom. (Pembimbing)
NIP. 197712172003121001

Waskitho Wibisono, S.Kom., M.Eng., Ph.D. (Penguji 1)
NIP. 197410222000031001

Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. (Penguji 2)
NIP. 198410162008121002

Hadziq Fabroyir, S.Kom., Ph.D. (Penguji 3)
NIP. 051100125

Dekan Fakultas Teknologi Informasi dan Komunikasi,

Dr. Agus Zainal Arifin, S.Kom, M.Kom
NIP. 19720809 199512 1 001

(Halaman sengaja dikosongkan)

PERNYATAAN KEASLIAN

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tesis saya dengan judul:

NON-PLAYABLE CHARACTER ADAPTIF PADA GAME RPG MENGGUNAKAN METODE LOGARITHMIC LEARNING FOR GENERALIZED CLASSIFIER NEURAL NETWORK

adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pusaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai dengan peraturan yang berlaku.

Surabaya, 28 Januari 2019

Izza Mabruroh

NRP: 5116201012

(*Halaman sengaja dikosongkan)*

NON-PLAYABLE CHARACTER ADAPTIF PADA *GAME* RPG MENGGUNAKAN METODE LOGARITHMIC LEARNING FOR GENERALIZED CLASSIFIER *NEURAL NETWORK*

Nama Mahasiswa : Izza Mabruroh

NRP : 5116201012

Pembimbing : Dr. Eng. Darlis HeruMurti, S.Kom, M.Kom

ABSTRAK

Non-playable Character (NPC) merupakan salah satu karakter yang penting dalam *game*. NPC yang bersifat otonom dan adaptif, dapat menyesuaikan aksi dengan aksi pemain dan keadaan lingkungan. Untuk menentukan aksi NPC, peneliti sebelumnya menggunakan metode *Neural Network* namun terdapat kelemahan yakni aksi yang dihasilkan tidak sesuai dengan yang diinginkan sehingga akurasinya kurang baik. Penelitian ini mengatasi masalah akurasi yang kurang baik dengan menggunakan metode *Logarithmic Learning for Generalized Classifier Neural Network* (L-GCNN) dengan menggunakan 6 parameter *input* yakni kesehatan NPC, jarak dengan pemain, NPC lain terlibat atau tidak, daya serang, jumlah NPC dan level NPC. Sedangkan *output*nya yakni menyerang sendiri, menyerang berkelompok dan menjauh.

Untuk pengujian, penelitian ini diuji cobakan pada *game* RPG. Dari hasil uji coba yang dilakukan menunjukkan bahwa metode L-GCNN memiliki akurasi yang lebih baik dari 3 metode yang dibandingkan yakni 7% lebih baik dari NN dan SVM serta lebih baik 8% dari RBFNN karena pada metode L-GCNN terdapat proses enkapsulasi yakni data yang mempunyai kelas yang sama akan dikelompokkan menjadi satu. Sedangkan untuk waktu *training* L-GCNN lebih lama 30% dari metode NN karena pada L-GCNN satu neuron terdiri dari satu data dimana pada NN lebih sedikit neuron pada *hidden layer*.

Kata kunci : *Non-playable character* (NPC), L-GCNN, NPC adaptif

(***Halaman ini sengaja dikosongkan***)

Adaptive non-playable character in RPG *game* using logarithmic learning for generalized classifier *Neural Network* method

Name : Izza Mabruroh

NRP : 5116201012

Supervisor : Dr. Eng. Darlis HeruMurti, S.Kom, M.Kom

ABSTRACT

Non-playable Character (NPC) is one of the important characters in the game. NPCs that are autonomous and adaptive, can adjust actions with player actions and environmental conditions. To determine the actions of the NPC, the previous researchers used the Neural Network method but there were weaknesses, namely the action produced was not in accordance with the desired so the accuracy was not good. This study overcomes the problem of accuracy that is not good by using the Logarithmic Learning for Generalized Classifier Neural Network (L-GCNN) method using 6 input parameters namely NPC health, distance with players, other NPCs involved or not, attack power, number of NPCs and NPC levels . While the output is single attack, keep the distance and attacking in group.

For testing, this study was tested on RPG games. From the results of the experiments conducted shows that the L-GCNN method has better accuracy than the 3 methods compared to 7% better than NN and SVM and 8% better than RBFNN because in the L-GCNN method there is an encapsulation process that is data having the same class will be grouped together. Whereas the L-GCNN training time is 30% longer than the NN method because on L-GCNN one neuron consists of one data where in NN there are fewer neurons in the hidden layer.

Keyword : *Non-playable character* (NPC), Finite state Machine (FSM), L-GCNN, Adaptive NPC

(*Halaman ini sengaja dikosongkan)*

KATA PENGANTAR

بسم الله الرحمن الرحيم

Segala puji bagi Allah SWT yang telah melimpahkan rahmat serta karuniaNya kepada penulis sehingga bisa menyelesaikan tesis dengan judul "Non-Playable Character adaptif pada *game* RPG menggunakan metode Logarithmic Learning for Generalized Classifier *Neural Network* (LGCNN)" dengan baik.

Shalawat serta salam semoga tercurah kepada Nabi Agung Muhammad SAW yang telah membimbing umatnya dari gelapnya kekufuran menuju cahaya Islam yang terang benderang.

Penulis menyadari keterbatasan pengetahuan yang penulis miliki, karena itu tanpa keterlibatan dan sumbangsih dari berbagai pihak, sulit bagi penulis untuk menyelesaikan thesis ini. Maka dari itu dengan segenap kerendahan hati patutlah penulis ucapkan terima kasih kepada:

1. Allah SWT atas limpahan rahmat-Nya sehingga penulis dapat menyelesaikan buku Tesis ini dengan baik.
2. Kedua orang tua penulis, suami dan anak yang selalu memotivasi dan selalu mendoaakan yang agar dilancarkan dan dimudahkan dalam menyelesaikan tesis ini.
3. Dr. Eng. Darlis Herumurti, S.Kom, M.Kom. selaku dosen pembimbing yang telah mengarahkan, memberi koreksi, dan motivasi dalam tesis ini..
4. Bapak Dewan penguji selaku dosen penguji yang telah memberikan saran dan kritik dalam tesis ini.
5. Bapak Waskitho Wibisono, S.Kom., M.Eng., PhD selaku ketua program pascasarjana Teknik Informatika ITS, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
6. Kepada teman-teman seperjuangan S2 Teknik Informatika ITS.
7. Semua pihak yang tidak mungkin penulis sebutkan satu-persatu, atas segala yang telah diberikan kepada penulis dan dapat menjadi pelajaran.

Sebagai penutup, penulis menyadari dalam tesis ini masih banyak kekurangan dan jauh dari sempurna. Semoga apa yang menjadi kekurangan bisa disempurnakan oleh peneliti selanjutnya. Apa yang menjadi harapan penulis, semoga karya ini bermanfaat bagi kita semua. Amin.

Surabaya, Januari 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

(***Halaman ini sengaja dikosongkan***)

DAFTAR TABEL

(***Halaman ini sengaja dikosongkan***)

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Sejak lahirnya ide kecerdasan buatan, *game* merupakan salah satu item yang membantu kemajuan penelitian AI. *Game* tidak hanya menimbulkan masalah yang menarik dan kompleks bagi AI untuk dipecahkan, *game* juga menyediakan lahan untuk kreatifitas dan ekspresi bagi pengguna. Dengan demikian bisa dibilang bahwa *game* merupakan domain langka dimana terdapat unsur seni dan interaksi yang menjadikan *game* menjadi unik dan favorit untuk studi AI. Tapi bukan hanya AI yang maju melalui penelitian, *game* juga telah maju melalui penelitian AI.

Salah satu karakter dalam *game* adalah *Non-Playable Character* (NPC) dimana menjadi domain yang semakin menantang untuk teknik kecerdasan buatan karena sifatnya yang komplek. Keberadaan NPC dapat menciptakan suatu agen cerdas yang realistis. Jika tidak ada NPC yang cerdas dimana NPC tidak bersifat otonom dan adapatif yakni NPC dapat melakukan perubahan perilaku menyesuaikan aksi pemain dan kedaaan *environment* tanpa dikontrol pemain, maka *game* dilihat dari sisi *gameplay* dapat dikatakan tidak menarik dan cenderung membosankan.

Salah satu metode yang umum digunakan untuk mengatur perilaku NPC adalah *Finite State Machine* dimana metode ini merupakan metode sederhana yang mudah untuk diimplementasikan, mudah diprediksi responnya, fleksibel dan mempunyai komputasi yang ringan. Namun mempunyai kelemahan yakni kondisi untuk transisi *state* yang tetap dan kurang tepat digunakan dalam *game* karena sifatnya yang bisa diprediksi.

Penelitian tentang metode FSM dalam *game* telah dilakukan oleh Edmond dkk dimana dalam penelitian ini, FSM digunakan untuk mengontrol pegulat selama *game* dimainkan dalam simulasi aksi menyerang dan bertahan. Penelitian ini menunjukkan bahwa FSM dapat mensimulasikan interaksi antar pegulat secara realistis dan *realtime* (Edmond & toku,2011). Peneliti selanjutnya menggunakan FSM pada *game* simulasi "pertolongan pertama" untuk memodelkan tiap

penanganan yang harus dilakukan. Penggunaan FSM dalam penelitian ini dapat membuat pemain lebih paham secara mendalam tentang pengetahuan dasar penanganan pertolongan pertama (Rosalina dkk,2015).

Pemodelan perilaku NPC menggunakan FSM juga dilakukan oleh Tito dan Hanny pada *game* pengenalan unsur kimia. FSM digunakan untuk menentukan jenis musuh, perilaku musuh dan tingkat kesulitan *game*. Model perilaku musuh tersebut akan digunakan utnuk perhitungan pemodelan selanjutnya dan untuk merubah performa objek-objek yang ada dalam *game* (Tito & Hanny,2016). Dikarenakan FSM mempunyai kelemahan yakni sifatnya yang bisa diprediksi, untuk mengatasi kelemahan ini peneliti lain menambahkan FSM ini dengan metode yang lain misalnya *fuzzy*.

Yunifa dkk, menerapkan logika *fuzzy* dan *Hierarchical Finite State Machine* (HFSM) untuk mengatur perilaku NPC agar dapat meniru strategi manusia dalam *game* peperangan. Hasil yang dicapai antara lain HFSM berhasil memodelkan perilaku manuver tiap NCP dan penerapan *fuzzy* untuk mengatur perilaku NPC berhasil mengungguli NPC tanpa *fuzzy* hingga 80% (Nugroho, 2011). Selanjutnya Supeno dkk melakukan penelitian pada *game* close combat menggunakan kordinator *fuzzy* dimana *rule base* digunakan untuk mengatur aksi bertarung sebuah NPC. Aksi kordinasi dari penelitian ini tidak cukup menghasilkan aksi yang bervariasi dan namun terkadang aksi tidak sesuai yang diinginkan. (supeno dkk,2013).

Penelitian selanjutnya menggunakan gabungan dari metode FSM, *Fuzzy* (FuSM) dan *Rule based System* untuk mengatur perilaku otonom dan adaptif NPC pada *game* 3 dimensi. Penelitian ini menghasilkan tingkat kehalusan pergerakkan NPC yang lebih tinggi, pemrosesan grafik yang lebih cepat, pemrosesan *main thread* atau kinerja performance yang lebih cepat dan proses renderer komponen *game* yang cukup cepat (Fahrul dkk, 2014).

Berdasarkan penelitian yang sudah ada, penambahan *fuzzy* dalam FSM atau yang umum disebut *Fuzzy State Machine* kurang optimal yakni masih menghasilkan aksi yang dapat diprediksi dikarenakan penggunaan *rule base* yang ada dalam *fuzzy* mempunyai *output* yang sudah pasti atau jelas.

Andreas, menggunakan jaringan saraf tiruan (*Neural Network*) dengan 3 *layer* untuk mengontrol perilaku agen AI pada *game* RTS (*Real Time Strategy*), ANN dipilih karena kemampuanya untuk mengeneralisasi berbagai keadaan. Penelitian ini menghasilkan agen AI yang adaptif yang dapat bereaksi terhadap perubahan keadaan lingkungan dan aksi tidak bisa diprediksi. Kelemahan dari penelitian ini adalah tingkat akurasi rendah dilihat dari aksi yang dihasilkan terkadang tidak sesuai sehingga memiliki akurasi yang kurang baik (andreas, 2009). Peneliti selanjutnya menggunakan jaringan saraf tiruan pada permainan catur yang adaptif untuk menentukan perpindahan pemain yang paling efektif. Hasil dari penelitian ini adalah sistem dapat menentukan perpindahan untuk mendapatkan winning position dalam permainan catur (Diwas, 2013).

Buse melis, mengusulkan pengembangan dari *Neural Network* menggunakan fungsi radial basis (RBFNN) yakni GCNN (*Generalized Classifier Neural Network*) untuk melakukan klasifikasi di beberapa data set dan terbukti memiliki performa lebih baik dari GRNN dan PNN, namun metode ini membutuhkan memori yang besar. (Buse Melis, 2013). Peneliti sebelumnya mengatasi kekurangan dari GCNN dengan menambahkan fungsi logarithmic (L-GCNN) untuk pengoptimalan smoothing parameter sehingga dapat mengurangi jumlah iterasi, mengurangi waktu *training* serta mendapat akurasi yang baik di banding metode sebelumnya. (Buse Melis, 2014).

Berdasarkan pemaparan diatas, peneliti mengusulkan untuk menggunakan metode *logaritmic learning for generalized classifier Neural Network* (L-GCNN) untuk mengatur perilaku adaptif tiap NPC dalam *game* dimana metode L-GCNN sesuai literatur yang ada, merupakan metode yang paling baik dari beberapa algoritma NN yang diusulkan peneliti lain karena metode ini memiliki performa yang lebih baik dari metode NN yang lain sehingga menghasilkan akurasi yang baik.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam penelitian ini adalah sebagai berikut :

1. Bagaimana mengatasi akurasi aksi yang kurang baik yang dihasilkan pada penelitian menggunakan *Neural Network* dengan menggunakan metode *Logarithmic Learning for Generalized Classifier Neural Network*?
2. Bagaimana menjadikan NPC mempunyai sifat adaptif sesuai dengan keadaan NPC, pemain dan lingkungan?

1.3. Tujuan

Tujuan dari penelitian ini adalah menghasilkan NPC yang berperilaku adaptif sesuai dengan keadaan NPC, aksi pemain serta lingkungannya dengan mengimplementasikan metode *Logarithmic Learning for Generalized Classifier Neural Network* (L-GCNN) untuk menentukan keputusan *state* yang aktif serta aksi NPC dari variabel yang sudah ditentukan.

1.4. Kontribusi Penelitian

Kontribusi dari penelitian ini adalah mengatasi akurasi aksi yang kurang baik dari metode *Neural Network* dengan mengimplementasikan metode *Logarithmic Learning for Generalized Classifier Neural Network* (L-GCNN) sebagai metode pengambil keputusan *state* yang aktif sehingga memiliki akurasi aksi yang lebih baik.

1.5. Batasan Masalah

Pada penelitian ini terdapat beberapa batasan masalah antara lain :

1. Metode digunakan pada permainan berjenis *role playing game* (RPG) dalam satu *scene* permainan.
2. NPC berjenis *enemies* dengan tipe *interacted NPC*.
3. Uji coba dilakukan dengan membandingkan antara metode L-GCNN dengan metode Multi *Layer* Perceptron (MLP), Radial Basis Function *Neural Network* (RBFNN) dan Support Vector Machine (SVM).
4. Data *training* didapat dari pemilihan keadaan dan aksi agen/NPC yang mungkin yang diteliti secara manual sesuai dengan nilai tiap parameter.

BAB II
KAJIAN PUSTAKA

Pada bab 2 ini akan dibahas dasar-dasar teori tentang metode yang digunakan, teori-teori penunjang serta penelitian-penelitian terkait yang membahas tentang adaptif NPC dalam *game*.

2.1. RPG *Game*

Game adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah. Teori permainan pertama kali ditemukan oleh sekelompok ahli Matematika pada tahun 1944. Teori itu dikemukakan oleh John von Neumann and Oskar Morgenstern yang berisi:"Permainan terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua sampai beberapa orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri atau pun untuk meminimalkan kemenangan lawan.

Game terdiri dari beberapa jenis, antara lain :

a. RPG *Game* (Role Playing *Game*)

   Game ini memiliki unsur yang unik, karena biasanya tidak ada tamat dalam *game*, kalaupun tamat hanya ceritanya aja dan masih bisa levelling. Pada *game* ini pemain akan menjalankan sebuah karakter yang bisa dipilih sendiri, mencari uang, membangun koneksi dengan NPC dan sebagainya. Gambar 2.1 merupakan contoh dari *game* RPG.

Gambar 2.1 Contoh RPG *game*

b. RTS *Game* (Real Time Strategy)

Game ini biasanya bersifat turn based ataupun bisa dimainkan secara bersamaan, identik dengan bermain melawan *human intelligent* (manusia vs manusia) yang biasanya menggunakan LAN ataupun internet. *Game* ini bisa dikatakan *game* perang-perangan contohnya DOTA, StarCraft, civilization, COC. Gambar 2.2 merupakan contoh dari *game* berenis RTS.

Gambar 2.2 Contoh RTS *game*

c. FPS *Game* (First Person Shooter)

Game ini merupakan *game* tembak-tembakan, pukul-pukulan yang menggunakan sudut pandang orang pertama. Biasanya pemain bisa melihat tangan dari karakter yang dimainkan. Contohnya Counter Strike, Far Cry 3, Call of Duty. Gambar 2.3 merupakan contoh dari *game* berjenis FPS.

Gambar 2.3 Contoh FPS *Game*

2.2. Non-Playable Character (NPC)

Non playable Character atau disebut NPC adalah karakter dalam *game* yang tidak dikontrol oleh pemain. NPC dapat berupa agen cerdas yang dibuat dengan menanamkan kecerdasan buatan dan memiliki perilaku yang mirip manusia seperti mengubah pola pikirnya dalam menanggapi tindakan lawan (Umarov dkk, 2012). NPC juga dapat dikatakan believable yang berarti memiliki kemiripan yang sangat tinggi terhadap karakter di *game* dengan yang ada di dunia nyata sehingga mampu membuat *game* lebih *immersive* (pemain akan menjadi seakan terlibat dalam permainan) (Nugroho dkk, 2011).

NPC sendiri dapat di bagi menjadi beberapa tipe.

- **Normal/Neutral NPC :** NPC yang digunakan hanya sebagai penambah latar dari *game* itu sediri. Biasanya NPC ini menjelaskan keadaan dari situasi atau kondisi pada latar *scene* pada *game*.
- **Dynamic NPC :** Berbeda dengan nomal NPC, NPC ini lebih berperan dalam *game*. NPC dengan tipe ini digunakan sebagai karakter tambahan dalam *game*. Biasanya dynamic NPC memiliki linear sendiri dan terkesan hidup dalam permainan.

Interacted NPC : Interacted NPC hamper sama dengan Dynamic NPC. Perbedaannya adalah interacted NPC memiliki variable tertentu yang biasanya di nyatakan sebagai "Status". Hal ini menyebabkan NPC ini dapat berubah-ubah sesuai dengan statusnya.

- **Dependent NPC :** Adalah NPC yang tidak bisa berinteraksi dengan pemain melainkan dengan Choices atau tindakan pemain.
- **Side NPC :** Side NPC merupakan NPC yang benar-benar tidak bisa berinteraksi. Biasanya digunakan hanya sebagai pelengkap latar saja.
- **Shopper :** Dalam *game* bergenre RPG atau MMORPG, biasanya terdapat NPC shopper. NPC ini bertindak sebagai pedagang barang yang dapat ditransaksikan ke pemain.

NPC dikategorikan menjadi 3 kategori yakni *enemie*, *partner* dan *support character* (Laird, 2000). *Enemie*/ musuh adalah karakter yang menentang pemain dengan menyerang pemain menggunakan senjata atau yang lainnya. *Partner* /

Mitra mengambil peran berlawanan dan berusaha melindungi pemain. Sebagai alternatif, karakter mitra bisa melakukan tugasseperti menjual barang dan dalam beberapa permainan, karakter mitra bisa diajarkan untuk melakukan perilaku tertentu oleh pemain. Selanjutnya *Support Character* / karakter pendukung merupakan karakter yang mendukung alur cerita permainan dengan menawarkan pencarian, saran, barang untuk dijual atau pelatihan. Dalam penelitian ini menggunakan NPC yang berjenis *enemies* dengan tipe dinamis dan interkatif.

2.3. Perilaku Agen Otonom

NPC atau Non Playable Character adalah karakter dalam *game* yang perilakunya tidak dikontrol oleh manusia/pemain. Perilaku NPC dibagi menjadi 3, yaitu strategis, taktik dan reaktif . Perilaku stategi digunakan untuk mencapai tujuan jangka panjang, misalnya mengamankan wilayahnya. Setiap NPC selalu memiliki tujuan jangka panjang dan jangka pendek. Perilaku taktis digunakan untuk mencapai tujuan jangka pendek yang lebih spesifik lagi. Sedangkan perilaku reaktif adalah reaksi sederhana sesuai dengan persepsi audio visualnya pada saat itu, seperti melompat, berjalan, membidik atau menembak.

Agen otonom harus mampu beraksi sesuai dengan sensor yang dia terima. Artinya agen tidak hanya berada dan menjadi bagian dari sebuah lingkungan buatan, tetapi menjadi pasangan dari lingkungan buatan itu. Arsitektur dan mekanisme agen harus terhubung dengan lingkungannya sehingga dapat merasakan setiap tujuan yang dirancang untuknya dan beraksi untuk memenuhi tujuan itu. (Maturana 1975, Maturana dkk., 1980, Varela 1991).

Setiap agen dalam *game* selalu memiliki tujuan (*goal*), seperti : menyerang musuh, tetap hidup, menangkap avatar. Agen atau aotonomous NPC juga mampu merasakan (*sensing*) perubahan pada lingkungannya, seperti kemampuan melihat halangan, mendengarkan suara. Dan agen juga dapat bertindak (*acting*) sesuai perubahan lingkungan yang ditemuinya, seperti : melompat untuk menghindari halangan, makan. agen diberi semacam indera untuk dapat mengindera lingkungannya dan untuk berkomunikasi dengan agen lain

2.4. Perilaku Adaptif

Perilaku adaptif merupakan penyesuaian perilaku karakter NPC terhadap NPC yang lain serta pemain dan lingkungan *game*. Pada saat NPC lain atau

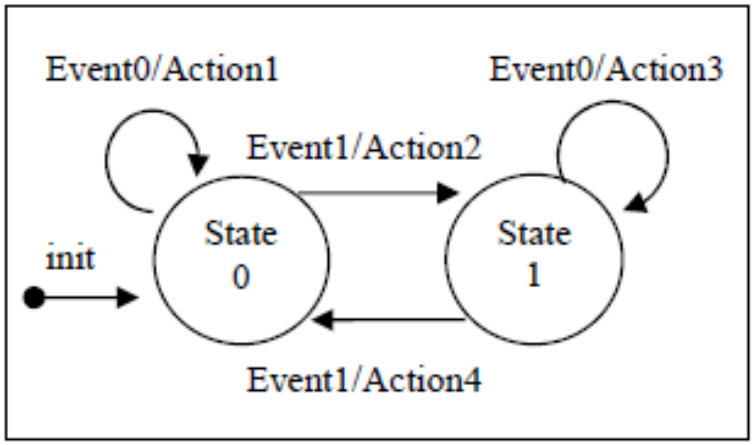
pemain melakukan suatu aksi, misalnya pemain menyerang NPC maka NPC akan melakukan reaksi atau menanggapi respon terhadap aksi pemain yaitu menyerang balik atau menghindar. Dalam penelitian ini, NPC diharapkan mempunyai perilaku adaptif yakni menyerang sendiri, menyerang secara berkelompok atau menghindar sesuai dengan variabel yang sudah ditentukan.

2.5. Finite State Machine (FSM)

Dalam perancangan kecerdasan buatan untuk *game*, FSM merupakan teknik yang paling banyak digunakan untuk permasalahan "decision making" dan sekaligus dengan scriptingnya juga digunakan secara luas untuk merancang sistem pengambil keputusan dalam *game*. Teknik ini digunakan untuk memodelkan perilaku sistem atau objek yang kompleks dengan kondisi yang sudah didefinisikan dalam satu set. Menurut Ian Millington (2009) dalam bukunya menyebutkan bahwa *Finite state machine* (FSM) masuk dalam ranah pembuat keputusan pada kecerdasan buatan.

Finite state machine (FSM) adalah sebuah metodologi perancangan sistem kontrol yang menggambarkan tingkah laku atau prinsip kerja sistem dengan menggunakan tiga hal yaitu : *State* (keadaan), *event* (kejadian) dan *action* (aksi). Pada satu saat sistem akan berada pada salah satu *state* yang aktif. Sistem dapat beralih atau bertransisi menuju *state* lain jika mendapatkan masukan atau *event* tertentu. Transisi keadaan ini umumnya disertai dengan aksi yang dilakukan oleh sistem ketika menanggapi masukan yang terjadi (Setiawan, 2006).

Gambar 2.4 menggambarkan FSM sederhana dengan dua buah *state* dan dua buah *input* serta empat buah *output*/*action.*



Gambar 2.4 Diagram FSM sederhana

Ketika sistem mulai dihidupkan, sistem akan bepindah menuju *state0* jika terdapat *event0/action1*, kemudian ketika terjadi *event1/action2*, maka *state0* akan berpindah ke *state1* dan seterusnya.

Dalam FSM *game* masing-masing aksi atau perilaku menempati satu state. Jadi selama karakter tetap dalam state tersebut, maka ia akan terus melakukan aksi yang sama. Jika permainan menentukan bahwa kondisi transisi terpenuhi, maka karakter berubah dari satu state ke state yang lain. Berdasarkan sifatnya, metode FSM ini sangat cocok digunakan sebagai basis perancangan perangkat lunak pengendalian yang bersifat reaktif dan real time. Salah satu keuntungan penggunaan FSM adalah kemampuannya untuk mendekomposisi aplikasi yang besar dengan menggunakan sejumlah item state yang kecil.

2.6. Machine Learning

Pembelajaran mesin atau lebih dikenal *Machine Learning* merupakan bidang ilmu komputer yang memberi sistem komputer kemampuan untuk belajar (yaitu meningkatkan kinerja secara progesif pada tugas tertentu) dengan data, tanpa diprogram secara eksplisit (Andreas, 2009). Teknik pembelajaran mesin berpotensi menghasilkan agen dengan kemampuan untuk beradaptasi dan belajar, sehingga menjaga *game* tetap menarik lebih lama (Miikkulainen, 2006). Sebagian besar teknik pembelajaran mesin seperti jaringan saraf tiruan lebih fleksibel dan bisa lebih mudah digunakan kembali dari pada menggunakan *finite state machine* yang komplek yang harus dibangun dari nol untuk tiap permainan. Pembelajaran

mesin dapat membuat *game* lebih menarik dan mengurangi biaya produksi (Fogel, 2004).

Oleh karena itu, dalam penelitian ini akan menggunakan salah satu teknik pembelajaran mesin yakni pengembangan dari jaringan saraf tiruan (*Neural Network*) untuk melakukan pembelajaran pada tiap NPC dengan tujuan menghasilkan *behaviour* yang berbeda untuk tiap NPC sesuai dengan *input*an yang ada dalam membentuk suatu tim.

Machine learning terdiri dari 2 macam pembelajaran yakni pembelajaran *offline* dan pembelajaran *online*. Pembelajaran *offline* mengacu pada pembelajaran yang dilakukan tanpa pemain yang bermain *game*. Data sampel digunakan untuk melatih agen *game*. Dalam penelitian ini pembelajaran *offline* digunakan untuk mendapatkan data sampel/ *training* dasar yang digunakan untuk proses *training* dalam metode yang digunakan.

2.7. Artificial *Neural Network*

Otak manusia merupakan organ terpenting dan memiliki struktur yang sangat kompleks terdiri dari neuron-neuron serta penghubungnya yang disebut sinapsis. Neuron akan bekerja berdasarkan impuls atau sinyal yang diberikan padanya dan meneruskannya pada neuron lain.

Jaringan saraf tiruan atau yang umum disebut *Neural Network* merupakan jaringan dari sekelompok unit pemroses kecil yang di modelkan berdasarkan sistem saraf manusia. Jaringan Syaraf Tiruan (JST) adalah sistem komputerisasi sebagai pemroses informasi yang memiliki karater mirip dengan jaringan syaraf biologi pada saat menangkap informasi dari ‘dunia luar’. Maksud sebenarnya dari JST adalah berusaha membuat sebuah model sistem komputasi informasi yang dapat menirukan rangkaian cara kerja jaringan syaraf biologis.

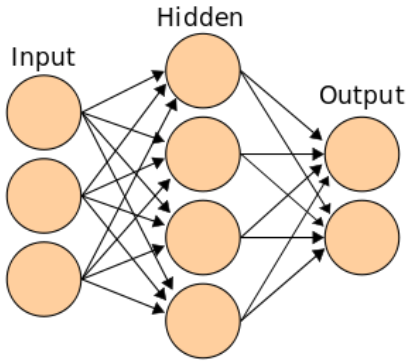
Model jaringan syaraf ditunjukkan dengan kemampuannya dalam emulasi, analisis, prediksi dan asosiasi. Kemampuan yang dimiliki jaringan syaraf tiruan dapat digunakan untuk belajar dan menghasilkan aturan atau operasi dari beberapa contoh atau *input* yang dimasukkan dan membuat prediksi tentang kemungkinan *output* yang akan muncul atau menyimpan karakteristik *input* yang diberikan kepada jaringan syaraf tiruan. Salah satu organisasi yang sering digunakan dalam

paradigma jaringan syaraf tiruan adalah perambatan galat mundur atau *backpropagation.* (Hermawan, 2006)

Jaringan Syaraf Tiruan ditentukan oleh 3 hal:

a. Arsitektur jaringan, sebagai pola hubungan antar neuron.
b. Algoritma, merupakan metode untuk menentukan bobot penghubung atau bisa juga disebut proses *training*/learning.
c. Fungsi aktivasi

.JST dapat digunakan untuk memodelkan hubungan yang kompleks antara *input* dan *output* untuk menemukan pola-pola pada data. Gambar 2.5 merupakan arsitektur *layer Neural Network*.



Gambar 2.5 Arsitektur *layer* NN

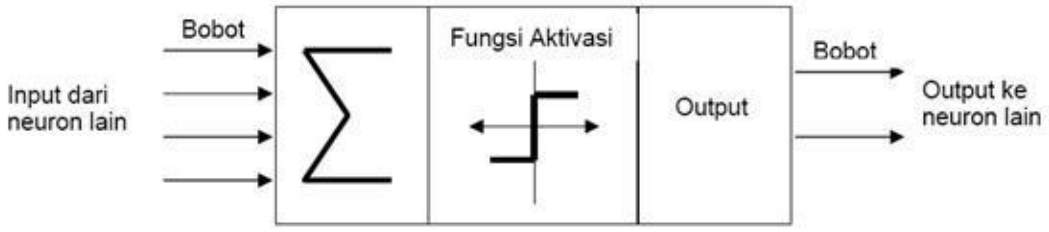
Secara umum, lapisan pada JST dibagi menjadi tiga bagian:

- Lapis masukan (*input layer*) terdiri dari *neuron* yang menerima data masukan dari variabel X. Semua *neuron* pada lapis ini dapat terhubung ke *neuron* pada lapisan tersembunyi atau langsung ke lapisan luaran jika jaringan tidak menggunakan lapisan tersembunyi.
- Lapisan tersembunyi (*hidden layer*) terdiri dari *neuron* yang menerima data dari lapisan masukan.
- Lapisan luaran (*output layer*) terdiri dari *neuron* yang menerima data dari lapisan tersembunyi atau langsung dari lapisan masukan yang nilai luarannya melambangkan hasil kalkulasi dari X menjadi nilai Y.

Fungsi dari *Neural Network* diantaranya adalah :

1. Pengklasifikasi pola
2. Memetakan pola yang didapat dari *input* ke dalam poal baru pada *output*
3. Penyimpanan pola yang akan dipanggil kembali
4. Memetakan pola-pola yang sejenis
5. Pengoptimasian permasalahan
6. Prediksi

Karakteristik dari ANN dilihat dari pola hubungan antar neuron, metode penentuan bobot dari tiap koneksi dan fungsi aktifasinya. Gambar 2.6 menjelaskan struktur ANN secara mendasar.



Gambar 2.6 Struktur ANN

Neural Network dibangun dari banyak node/unit yang dihubungkan oleh link secara langsung. Link dari unit yang satu ke unit yang lainnya digunakan untuk melakukan propagasi aktivasi dari unit pertama ke unit selanjutnya. Setiap link memiliki bobot numerik. Bobot ini menentukan kekuatan serta penanda dari sebuah konektivitas.

Proses pada ANN dimulai dari *input* yang diterima oleh neuron beserta dengan nilai bobot dari tiap-tiap *input* yang ada seperti gambar 2.6. Setelah masuk ke dalam neuron, nilai *input* yang ada akan dijumlahkan oleh suatu fungsi perambatan (*summing function*), yang bisa dilihat seperti pada di gambar dengan lambang sigma (∑). Hasil penjumlahan akan diproses oleh fungsi aktivasi setiap neuron, disini akan dibandingkan hasil penjumlahan dengan *threshold* (nilai ambang) tertentu. Jika nilai melebihi *threshold*, maka aktivasi neuron akan dibatalkan, sebaliknya, jika masih dibawah nilai *threshold*, neuron akan diaktifkan. Setelah aktif, neuron akan mengirimkan nilai *output* melalui bobot-

bobot *output*nya ke semua neuron yang berhubungan dengannya. Proses ini akan terus berulang pada *input-input* selanjutnya.

ANN terdiri dari banyak *neuron* di dalamnya. Neuron ini akan dikelompokkan ke dalam beberapa *layer*. Neuron yang terdapat pada tiap *layer* dihubungkan dengan neuron pada *layer* lainnya. Hal ini tentunya tidak berlaku pada *layer input* dan *output*, tapi hanya *layer* yang berada di antaranya. Informasi yang diterima di *layer input* dilanjutkan ke *layer-layer* dalam ANN secara satu persatu hingga mencapai *layer* terakhir/*layer output*. *Layer* yang terletak di antara *input* dan *output* disebut sebagai *hidden layer*. Namun, tidak semua ANN memiliki *hidden layer*, ada juga yang hanya terdapat *layer input* dan *output* saja.

Pelatihan perlu dilakukan pada suatu jaringan syaraf tiruan sebelum digunakan untuk menyelesaikan masalah. Hasil pelatihan jaringan syaraf tiruan dapat diperoleh tanggapan yang benar (yang diinginkan) terhadap masukan yang diberikan. Jaringan syaraf tiruan dapat memberikan tanggapan yang benar walaupun masukan yang diberikan terkena derau atau berubah oleh suatu keadaan. (Hermawan, 2006).

Dalam metode NN, parameter kuncinya adalah bobot. Bobot merupakan suatu nilai yang mendefinisikan tingkat atau kepentingan hubungan antara suatu node dengan node yang lain. Semakin besar bobot suatu hubungan menandakan semakin pentingnya hubungan kedua node tersebut. Bobot merupakan suatu hubungan berupa bilangan real maupun integer, tergantung dari jenis permasalahan dan model yang digunakan. Bobot-bobot tersebut bisa ditentukan untuk berada didalam interval tertentu. selama proses pelatihan, bobot tersebut dapat menyesuaikan dengan pola-pola *input*.

Jaringan dengan sendirinya akan memperbaiki diri terus-menerus karena adanya kemampuan untuk belajar. Setiap ada suatu masalah baru, jaringan dapat belajar dari masalah baru tersebut, yaitu dengan mengatur kembali nilai bobot untuk menyesuaikan karakter nilai. (Puspaningrum, 2006).

Pelatihan pada jaringan syaraf *backpropagation, feedfoward* (umpan maju) dilakukan dalam rangka perhitungan bobot sehingga pada akhir pelatihan akan diperoleh bobot-bobot yang baik. Selama proses pelatihan, bobot-bobot diatur secara iteratif untuk meminimumkan *error* (kesalahan) yang terjadi.

Error (kesalahan) dihitung berdasarkan rata-rata kuadrat kesalahan (MSE). Rata-rata kuadrat kesalahan juga dijadikan dasar perhitungan unjuk kerja fungsi aktivasi. Sebagian besar pelatihan untuk jaringan *feedfoward* (umpan maju) menggunakan gradien dari fungsi aktivasi untuk menentukan bagaimana mengatur bobot-bobot dalam rangka meminimumkan kinerja. Gradien ini ditentukan dengan menggunakan suatu teknik yang disebut *backpropagation.*

Algoritma pelatihan standar *backpropagation* akan menggerakkan bobot dengan arah gradient negative. Prinsip dasar dari algoritma *backpropagation* adalah memperbaiki bobot-bobot jaringan dengan arah yang membat fungsi aktifasi menjadi turun dengan cepat.

Pelatihan *backpropagation* meliputi 3 tahapan sebagai berikut.

1. Propagasi maju.

Pola masukan dihitung maju mulai dari *input layer* hingga *output layer* menggunakan fungsi aktifasi yang ditentukan.

2. Propagasi mundur.

Selisih antara keluaran jaringan dengan target yang diinginkan merupakan kesalahan yang terjadi. Kesalahan yang terjadi itu dipropagasi mundur. Dimulai dari garis yang berhubungan langsung dengan unit-unit di *output layer*.

3. Perubahan bobot.

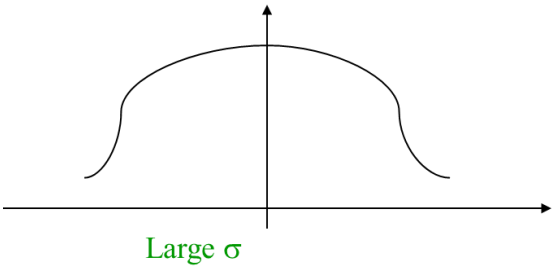
Modifikasi bobot untuk menurunkan kesalahan yang terjadi. Ketiga fase tersebut diulang-ulang terus hingga kondisi penghentian dipenuhi. (puspaningrum, 2006).

2.8. Logarithmic Learning for Generalized Classifier *Neural Network*

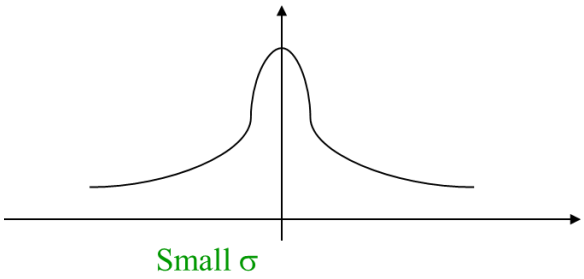
Salah satu teknik kecerdasan buatan yang sering digunakan untuk pengambilan keputusan dengan proses pembelajaran adalah *Neural Network*. Multi*layer* perceptrons (MLP), radial basis functions (RBF), probabilistic *Neural Network* (PNN), self organizing maps (SOM), cellular *Neural Network* (CNN), recurrent *Neural Network* and conic section funsction *Neural Network* (CSFNN) merupakan beberapa pengembangan *Neural Network*. Pada tahun 2013, peneliti bernama buse melis dan mutlu avci mengusulkan pengembangan lebih lanjut dari RBF-NN yang dinamai Generalized classifier *Neural Network* (GCNN). Berbeda

dengan standar RBF-NN yang terdiri dari 3 *layer* yakni *input*, hidden dan *output*, GCNN memiliki 5 *layer* yakni *input*, pattern, summation, normalization an *output*.

Berbeda dengan standar *Neural Network* yang menggunakan bobot sebagai parameter kunci, pada metode GCNN menggunakan parameter *smoothing* (σ) untuk menentukan performa dari metode. Namun metode GCNN dinilai membutuhkan memori yang besar serta kompleksitas waktu yang masih cukup lama. Oleh karena itu peneliti tersebut mengembangkan lagi pada tahun 2014 dengan menambahkan *logarithmic cost function* untuk pengoptimalan parameter *smoothing*. Gambar 2.7 dan 2.8 menunjukkan kurva smoothing parameter bagaimana memisah tiap data ketika nilainya tinggi dan rendah.



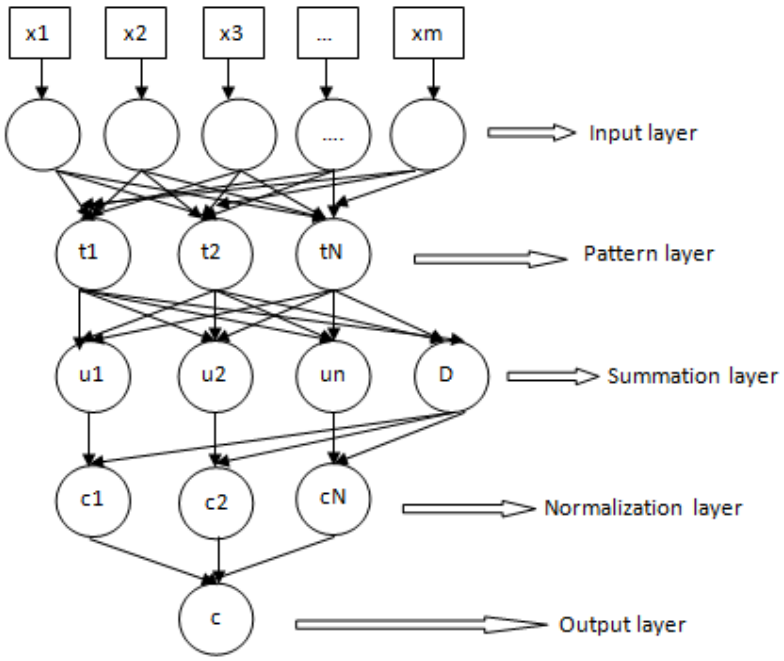
Gambar 2.7 Parameter smoothing tinggi



Gambar 2.8 Parameter smoothing rendah

2.8.1. Struktur L-GCNN

Struktur L-GCNN berbeda dengan struktur ANN yang memiliki 3 *layer* yaitu *input*, hidden dan *output*. Metode ini memiliki struktur 5 *layer* yaitu *input*, pattern, summation, normalization dan *output* seperti gambar 2.9.



Gambar 2.9 Struktur *layer* L-GCNN

Pada struktur L-GCNN gambar 2.9, *Input layer* mengirimkan vektor *input x* menuju ke pattern *layer*. Pattern *layer* terdiri atas satu neuron untuk tiap data *training*. Neuron pada pattern *layer* menghitung jarak euclidian antara vektor *input x* dan vektor data *training t*. *Output* pada pattern *layer* ditentukan dengan fungsi aktifasi RBF. Sebagai metode L-GCNN yang berbasis regresi, maka L-GCNN dibuat dengan struktur *one vs all discriminative*. Oleh karena itu, setiap data *training* memiliki N nilai yang ditentukan dengan apakan data tersebut termasuk kedalam suatu kelas atau tidak, jika data *training* termasuk kedalam *ith class* maka kelaske-*i* akan bernilai 0,9 dan kelas yang lainnya bernilai 0.1.

Summation *layer* memiliki neuron N+1 dimana N adalah jumlah keseluruhan kelas dan 1 merupakan neuron denominator. Pada *layer* ini, dilakukan perhitungan diverge effect term, numerator dan denominator. Ketikan neuron N menghitung jumlah perkalian dari diverge effect term dan *output* pattern *layer* (numerator) *ui*, neuron yang lain menghitung denominator. Normalization *layer* mencakup neuron yang mewakili tiap kelas. Kemudian keputusan pemenang dilakukan di *layer output* dengan memilih nilai *output* tertinggi pada normalization *layer*.

2.8.2. Gradient Descent *Training* Method

Merupakan algoritma optimasi orde pertama yang berulang. Tujuan dari metode ini adalah menemukan minimum fungsi yang dapat didiferensiasi dengan mengikuti arah gradien yang berlawanan dengan ukuran langkah yang ditentukan (Madsen, Nielsen,&Tingleff, 2004). Metode ini digunakan untuk optimasi nilai parameter *smoothing* seperti persamaan 2.1 dimana σ_{new} didapatkan dari parameter smoothing lama ditambahkan hasil kali antara learning rate dan delta cost function dibagi delta sigma.

$$\sigma_{new} = \sigma_{old} + lr * \frac{\partial e}{\partial \sigma} \quad (2.1)$$

Ket:

σ_{new} = parameter smoothing baru

σ_{old} = parameter smoothing lama

lr = learning rate

$\frac{\partial e}{\partial \sigma}$ = delta error/delta sigma

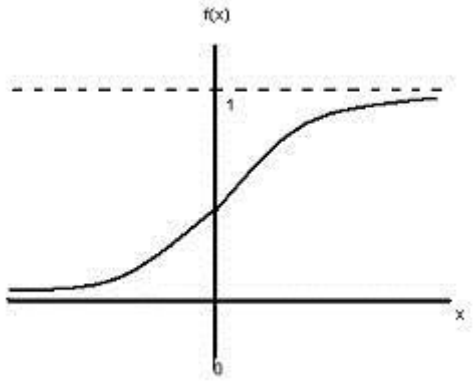
2.8.3. Fungsi Aktifasi

Merupakan operasi matematik yang digunakan pada sinyal *output*. Fungsi ini digunakan untuk mengaktifkan atau menonaktifkan neuron. Setiap neuron mempunyai keadaan internal yang disebut level aktivasi atau level aktivitas yang merupakan fungsi *input* yang diterima. Secara tipikal suatu neuron mengirimkan aktivitasnya kebeberapa neuron lain sebagai sinyal. Perilaku dari L-GCNN ditentukan oleh smoothing parameter dan *input*-intput fungsi aktifasi yang ditetapkan.

Ada beberapa pilihan fungsi aktivasi yang digunakan dalam metode L-GCNN, seperti fungsi sigmoid biner, dan sigmoid bipolar. Karakteristik yang harus dimiliki fungsi fungsi aktivasi tersebut adalah kontinue, diferensiabel, dan tidak menurun secara monoton. Fungsi aktivasi diharapkan dapat mendekati nilai-nilai maksimum dan minimum secara baik. Berikut ini adalah fungsi aktivasi yang sering digunakan yaitu: (Puspaningrum, 2006).

1. Fungsi Sigmoid Biner

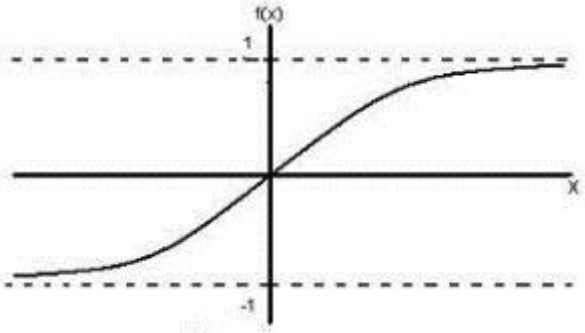
Fungsi ini digunakan untuk jaringan syaraf yang dilatih dengan menggunakan metode *backpropagation*. Fungsi sigmoid biner memiliki nilai pada range 0 sampai 1. Fungsi ini sering digunakan untuk jaringan syaraf yang membutuhkan nilai *output* yang terletak pada interval 0 sampai 1. Gambar 2,8 menunjukkan ilustrasi fungsi sigmoid biner dengan range (0,1).



Gambar 2.10 Ilustrasi fungsi sigmoid biner

2. Fungsi Sigmoid Bipolar

Fungsi sigmoid bipolar hampir sama dengan fungsi sigmoid biner, hanya saja *output* dari fungsi ini memiliki range antara 1 sampai -1. Gambar 2.9 menunjukkan ilustrasi fungsi sigmoid biner dengan range (-1,1).



Gambar 2.11 Ilustrasi fungsi sigmoid biner range 1, -1

Pada metode ini, digunakan fungsi aktifasi RBF berbasis gaussian menggunakan persamaan 2.2 dimana r merupakan jarak dari cluster center.

$$\varphi(r) = exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \sigma > 0 \qquad (2.2)$$

Ket :

r = jarak dari cluster pusat

σ = parameter smoothing

2.8.4. Perhitungan Rumus Tiap *Layer*

a) *Input layer*

Layer ini mengirimkan vektor *input x* ke pattern *layer*. Satu neuron mewakili tiap data *training*.

b) Pattern *layer*

Setelah data diterima dari *input layer*, kemudian menghitung jarak Euclidian antara vector *input x* dan vector data *training t* menggunakan persamaan 2.3 dimana dalam *layer* ini jumlah neuron sama dengan jumlah data *training*.

$$dist\ (j) = \left\| x - t_j \right\|^2, 1\ \leq j \leq p \qquad (2.3)$$

ket :

$dist\ (j) =$ jarak euclidian

x = vector *input* x

t_j = vektor data *training*

p = jumlah dari data *training*

output dari pattern *layer* ditentukan dengan menggunakan fungsi aktifasi RBF yakni gaussian menggunakan persamaan 2.4:

$$r(j) = exp\left(-1 * \frac{dist(j)}{2\sigma^2}\right), 1 \leq j \leq p \qquad (2.4)$$

ket :

$dist(j)$ = jarak euclidian

2σ = smoothing parameter

p = jumlah data *training*

Kemudian menentukan apakah data tersebut termasuk ke dalam kelas atau tidak seperti persamaan 2.5. Alasan pemilihan 0,9 dan 0,1 adalah untuk mencegah masalah stuck neuron pada proses pembelajaran. Jika data tersebut masuk ke dalam neuron kelas yang dituju maka diberi nilai 0.9 jika tidak maka akan diberi nilai 0.1.

$$y(j,i) = \begin{cases} 0.9 \\ 0.1 \end{cases} t_j \; menunjukkan \; kelas \; ith \; 1 \leq i \; \leq N \quad else \; 1 \leq j \leq p \tag{2.5}$$

Ket :

$y(j,i)$ = nilai *y* untuk data *training* ke-*j* dan kelas ke-*i*

c) Summation *layer*

Summation *layer* terdiri dari N+1 neuron dimana N merupakan total numlah kelas dan 1 neuron merupakan neuron denominator. Pada *layer* ini dilakukan perhitungan $d\,(j,i))$ dimana $d\,(j,i)$ menunjukkan nilai diverge effect term dari data *training* ke-*j* dan kelas ke -*i* seperti persamaan 2.6 dengan menggunakan fungsi eksponensial untuk meningkatkan efek dari *y(j,i).* Alasan penggunaan fungsi eksponensial adalah memberikan konvergensi terhadap kesalahan minimal dan mencegah overfitting. Dengan meningkatkan efek dari *y(j,i),* dapat memisahkan data dari kelas yang berbeda. $ymax$ diinisialisasi dengan nilai 0.9 yang menunjukkan nilai maksimum dari $y(j,i)$ dan diubah dengan nilai *output* maksimum tiap iterasi.

$$d\,(j,i) = \; exp^{(y(j,i)-ymax)} * y(j,i) \tag{2.6}$$

Ket :

$d\,(j,i) =$ diverge effect term

$ymax$ = nilai *y* maksimal

$y(j,i)$ = nilai *y* untuk data *training* ke-*j* dan kelas ke-*i*

Setelah itu menghitung numerator (u_i) dan denominator (D). Numerator dihitung pada neuron N menggunakan persamaan 2.7 dimana neuron N menghitung jumlah dari perkalian antara diverge effect term dan *output* dari *layer* pattern sedangkan denominator dihitung oleh neuron 1 pada summation *layer* menggunakan persamaan 2.8.

$$u_i \; = \sum_{j=1}^{p} d(j,i) * r(j) \tag{2.7}$$

$$D = \sum_{j=1}^{p} r(j) \quad (2.8)$$

Ket :

u_i = numerator untuk tiap kelas

$d(j,i)$ = diverge effect term

$r(j)$ = fungsi aktifasi RBF untuk tiap data *training*

D = Denominator

d) Normalization *layer*

Normalization *layer* terdiri dari N neuron yang mewakili tiap kelas. Tiap neuron membagi nilai numerator dengan denominator yang didapatkan dari *layer* sebelumnya. Seperti persamaan 2.9 dimana c_i menunjukkan *output* yang dinormalisasi dari *ith class.*

$$c_i = \frac{u_i}{D}, 1 \leq i \leq N \quad (2.9)$$

Ket :

c_i = *output* normalisasi

u_i = numerator

D = Denominator

e) *Output layer*

Akhirnya, mekanisme keputusan pemenang diberikan dengan persamaan 2.10, dimana *c* adalah vektor *output* pada normalization *layer, o*menunjukkan nilai pemenang neuron dan *id* menunjukkan kelas pemenang.

$$[o, id] = \max(c) \quad (2.10)$$

Setelah menemukan pemenang, kemudian menghitung *cost function (e)* menggunakan persamaan 2.11, dimana kalau standar GCNN menggunakan error kuadrat, sedangkan L-GCNN menggunakan *logarithmic*. $y(z, id)$ menunjukkan nilai *input* data *training* ke-z dan c_{id} menunjukkan kelas pemenang. Kemudian smoothing parameter diperbaharui dengan dengan persamaan 2.12 – 2.16 dimana lr menunjukkan nilai learning rate.

$$e = (y(z,id) * \log(c_{id})) + (1 - y(z,id)) * log(1 - c_{id}) \quad (2.11)$$

$$\sigma_{new} = \sigma_{old} + lr * \frac{\partial e}{\partial \sigma} \quad (2.12)$$

$$\frac{\partial e}{\partial \sigma} = y(z,id)\left(\frac{\frac{\partial c_{id}}{\partial \sigma}}{c_{id}}\right) + (1 - y(z,id)) * \left(\frac{\frac{-\partial c_{id}}{\partial \sigma}}{c_{id}}\right) \quad (2.13)$$

$$\frac{\partial c_{id}}{\partial \sigma} = \frac{b(id) - l(id) * c_{id}}{D} \quad (2.14)$$

$$b(id) = 2 * \sum_{j=1}^{p} d(j,id) * r(j) * \frac{dist(j)}{\sigma^3} \quad (2.15)$$

$$l(id) = 2 * \sum_{j=1}^{p} r(j) * \frac{dist\ (j)}{\sigma^3} \quad (2.16)$$

Ket :

e = cost function

$y(z,id)$ = nilai *y* dari data *training* pemenang ke –z

c_{id} = kelas pemenang

$\partial\sigma$ = delta parameter smoothing

∂e = delta cost function

$d(j,id)$ = nilai diverge effect term dari kelas pemenang

L-GCNN menggunakan pembelajaran online untuk mengurangi total cost. Jika nilai c_{id} berada diantara rentang nilai (0.1, 0.9) dan logarithmic dari c_{id} berada direntang (-2, -0.3), meskipun data *training* sampel sudah terklasifikasi dengan benar, smoothing parameter akan tetap diubah. Proses *training* berhenti jika sudah mencapai batas jumlah iterasi yang ditentukan. L-GCNN mengecek nilai parameter smoothing yang baru sebelum diupdate. Jika kurang dari 0 maka nilai parameter smoothing tidak diupdate

2.9. Penelitian Terkait

Untuk menunjang penelitian ini, dibutuhkan beberapa penelitian sebelumnya yang membahas tentang metode FSM dan L-GCNN. Tabel 2.1

merupakan beberapa penelitian terkait yang membahas tentang meode yang digunakan untuk mengatur aksi NPC.

Tabel 2.1 Penelitian Terkait

Judul penelitian	Metode yang digunakan dan hasil
A finite state machine based on topology coordinates for wrestling *game*s (2011)	- Menggunakan FSM berdasarkan topologi koordinat untuk mensimulasikan interaksi perilaku menyerang dan bertahan antara dua pegulat saat bertanding dimana karakter pegulat dikontrol secara manual oleh pemain. - Penelitian ini menunjukkan bahwa metodologi ini dapat mensimulasikan interaksi antar pegulat secara realistis untuk *game* adu gulat secara real time.
First aid simulation *game* with finite state machine model (2015)	- Menggunakan FSM pada *game* simulasi "pertolongan pertama" untuk memodelkan tiap penanganan yang harus dilakukan saat pertolongan pertama. FSM digunakan untuk mengontrol semua alur simulasi muali dari aturan sampai sistem menang kalah *game*. - Penggunaan FSM dalam penelitian ini dapat membuat pemain lebih paham secara mendalam tentang pengetahuan dasar untuk penanganan pertolongan pertama.
Pemodelan perilaku musuh menggunakan finite state machine pada *game* pengenalan unsur kimia (2016)	- Menggunakan FSM untuk menentukan jenis musuh, perilaku musuh dan tingkat kesulitan *game* - Hasil yang didapat, FSM dapat menghasilkan pemodelan perilaku musuh. Model perilaku musuh tersebut akan digunakan untuk perhitungan pemodelan selanjutnya dan untuk merubah performa objek-objek yang ada

	dalam *game*.
Desain perubahan perilaku pada NPC *game* Menggunakan logika fuzzy (2011)	- Menggunakan gabungan dari logika fuzzy dan Hierarchical Finite State Machine (HFSM) agar bisa membuat aksi dan reaksi otonom pada NPC pada *game* FPS - Berhasil memodelkan perilaku manuver tiap NPC dan terjadi respon perubahan perilaku terhadap kondisi yang dihadapi
Fuzzy Coordinator based intelligent agents for team coordination in close combat *game*s (2013)	- Menggunakan coordinator fuzzy pada *game* close combat dimana rulebase digunakan untuk mengatur aksi fighting sebuah NPC. - Tim NPC dengan coordinator fuzzy memiliki keunggulan kesehatan dibandingkan tanpa koordinasi fuzzy. Tim NPC dengan coordinator fuzzy dapat menjaga parameter kesehatan masing-masing maksimal 76% dan minimal 50%
Perilaku otonom dan adaptif non pemain character musuh pada *game* 3 dimensi menggunakan *Fuzzy state machine* dan *rule based system* (2014)	- Menggunakan kombinasi metode *Fuzzy, finite state machine* dan *rule based system* untuk mengembangkan variasi perilaku NPC musuh yang otonom dan adaptif - menghasilkan tingkat kehalusan pergerakkan NPC yang lebih tinggi, pemrosesan grafik yang lebih cepat, pemrosesan main thread atau kinerja performance yang lebih cepat dan proses renderer komponen *game* yang cukup cepat dibandingkan dengan yang hanya menggunakan 2 metode saja logika Fuzzy dan Finite State Machine (FuSM) atau yang hanya menggunakan 1 metode saja yaitu Rule Based System.

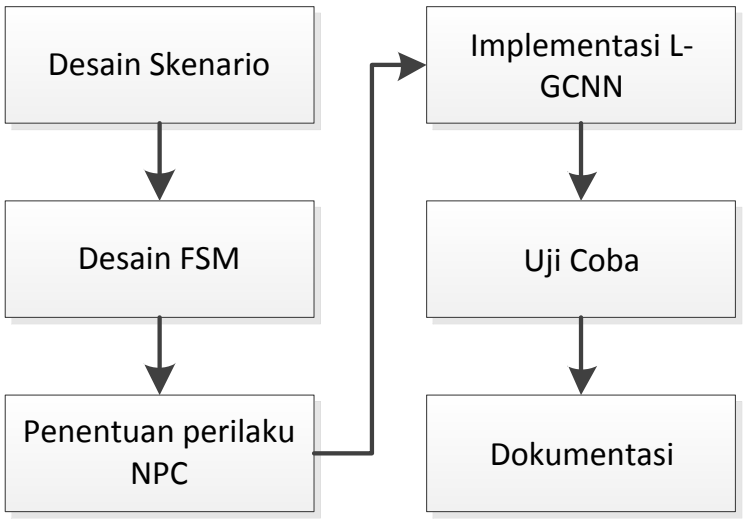
Creating adaptive *game* AI in a real time continous environment using *Neural Network* (2009)	- Menggunakan jaringan saraf tiruam dengan 3 *layer* untuk mengatur perilaku adaptive NPC yang terdiri dari 3 jenis pada *game* RTS - Hasil dari penelitian ini, agen AI mampu berperilaku adaptive sesuai *input*an yang ada namun membutuhkan komplek-sitas waktu yang lama dan terkadang aksi tidak sesuai harapan
Artificial *Neural Network* based adaptive chess playing machine (2013)	- Menggunakan jaringan saraf tiruan untuk menetukan perpindahan pemain yang paling efektif utnuk menemukan *winning position* dalam *game* catur yang adaptif. - Hasil dari penelitian ini, sistem yang dihasilkan menjadi sangat sederhana dan efektif karena merealisasikan dan mengisyaratkan langkah pemain untuk dipindahkan. -
Generalized classifier *Neural Network* (2013)	- Mengusulkan pengembangan dari NN yakni GCNN dengan 5 *layer* yakni *input*, pattern, summation, normalization dan *output* untuk melakukan klasifikasi dari beberapa data set - GCNN memiliki performa lebih baik dari pada standar NN, GRNN, PNN, MLP dan RBFNN namun mempunyai masalah pada memori yang besar
Logarithmic learning for generalized classifier *Neural Network* (2014)	- Mengatasi kelemahan penelitian sebelumnya dengan menambahkan fungsi logarithmic (L-GCNN) saat melakukan perhitungan cost function - Mengurangi jumlah iterasi, waktu *training*

	yang cepat dan akurasi yang lebih baik dari metode GCNN

(Halaman sengaja di kosongkan)

BAB III
METODOLOGI PENELITIAN

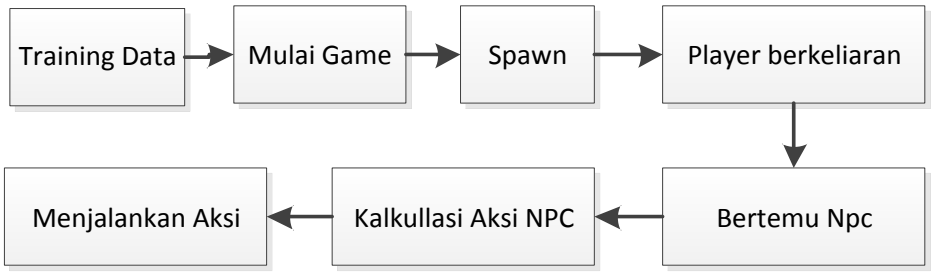
Pada penelitian ini, terdapat beberapa langkah dalam penerapan metode pada permainan. Gambar 3.1 merupakan diagram alur kerja yang digunakan pada penelitian ini dari perancangan desain skenario hingga pengujian metode. Mulai dari membuat desain scenario NPC dalam *game* kemudian membuat desain finite state machine untuk menentukan state apa saja yang ada dalam *game* serta mengatur aksi NPC. Selanjutnya menentukan perilaku apa saja yang akan dijadikan *output* aksi. Setelah perilaku ditentukan, selanjutnya mengimplementasikan L-GCNN untuk menentukan aksi NPC kemudian dilakukan pengujian dan yang terkahir dibuat dokumentasi berupa buku tesis.



Gambar 3.1 Skema Metodologi penelitian

3.1. Desain Skenario *Game*

Penelitian ini dilakukan pada *game* berjenis RPG dalam satu scene permainan dimana setelah *game* dimulai, semua karakter ditempatkan pada suatu titik yang ditentukan kemudian pemain akan bertemu musuh dalam penelitian ini NPC. Kemudian program menghitung dari *input*an NPC yang ada untuk menentukan behavior yang akan dilakukan. Setelah hasil didapatkan, NPC akan melakukan aksi sesuai hasil dari perhitungan dengan L-GCNN seperti gambar 3.2.



Gambar 3.2 Diagram alur permainan

Seperti yang tertera pada diagram alur (Gambar 3.2), sebelum permainan dimulai dilakukan proses *training* data terlebih dahulu, kemudian setelah proses *training* selesai dilakukan, pemain dan NPC spawning, spawning maksudnya adlah NPC dan Pemain ditempatkan dalam suatu titik kordinat atau dengan kata lain posisi awal. Ketika pemain berjalan dan bertemu dengan NPC, proses L-GCNN memulai kalkulasi berdasarkan hasil *training* awal untuk mendapatkan aksi untuk NPC. Setelah mendapatkan aksi yang diinginkan, aksi tersebut diaplikasikan pada NPC.

3.1.1. Desain Pemain

Gambar 3.3 Karakter pemain

Karakter pemain yang digunakan dapat dilihat pada Gambar 3.3. karakter pemain ini berupa karakter humanoid dimana control aksinya menggunakan keyboard untuk berjalan dan mouse untuk melakukan aksi. Control gerak pada

pemain ini menggunakan control basic dimana tombol W,A,S,D menjadi tombol untuk gerak pemain, tombol klik kiri pada mouse untuk melancarkan serangan pada musuh. Character pemain sendiri memiliki parameter untuk menentukan kesehatan dan kekuatannya serta posisi yang digunakan untuk pembanding.

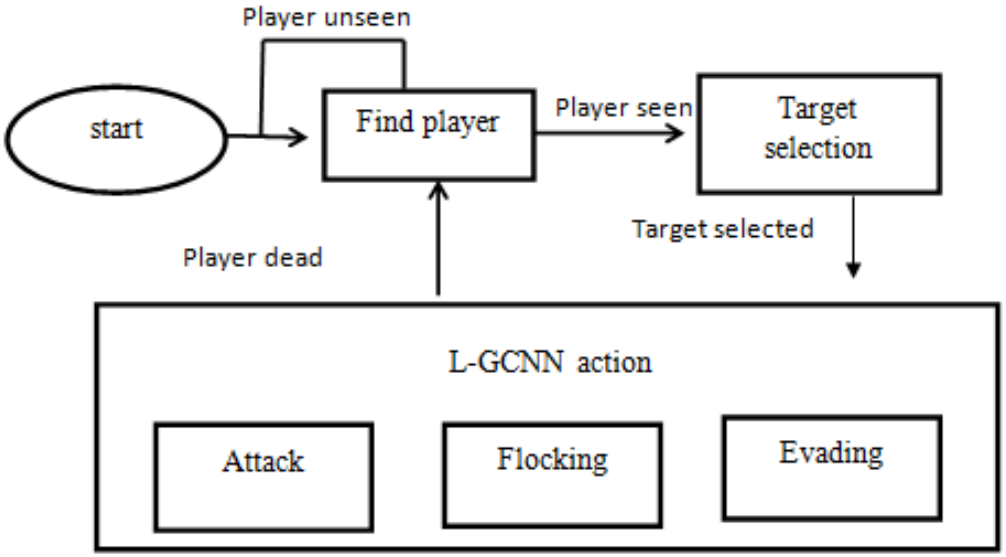
3.1.2. Desain NPC

Gambar 3.4 Karakter NPC

Karakter NPC yang disajikan pada Gambar 3.4 berupa karakter monster IMP bertipe humanoid dengan parameter NPC yang bisa diatur dan aksinya sesuai dengan *output* yang dihasilkan oleh metode yang diusulkan.

3.2. Desain FSM NPC

Penelitian ini mengembangkan metode *Finite State Machine* dengan menambahkan L-GCNN untuk mengambil keputusan *state* yang aktif seperti gambar 3.5, dimana ketika *game* dimulai, NPC belum menemukan pemain/musuh maka NPC akan terus mencari sampai terlihat, setelah musuh terlihat maka musuh akan di pilih dan NPC melakukan aksi yakni *single attack, single attack in grouping* atau *keep distance* sesuai dengan hasil perhitungan yang dilakukan oleh metode L-GCNN dengan mempertimbangkan *input* yang ada.



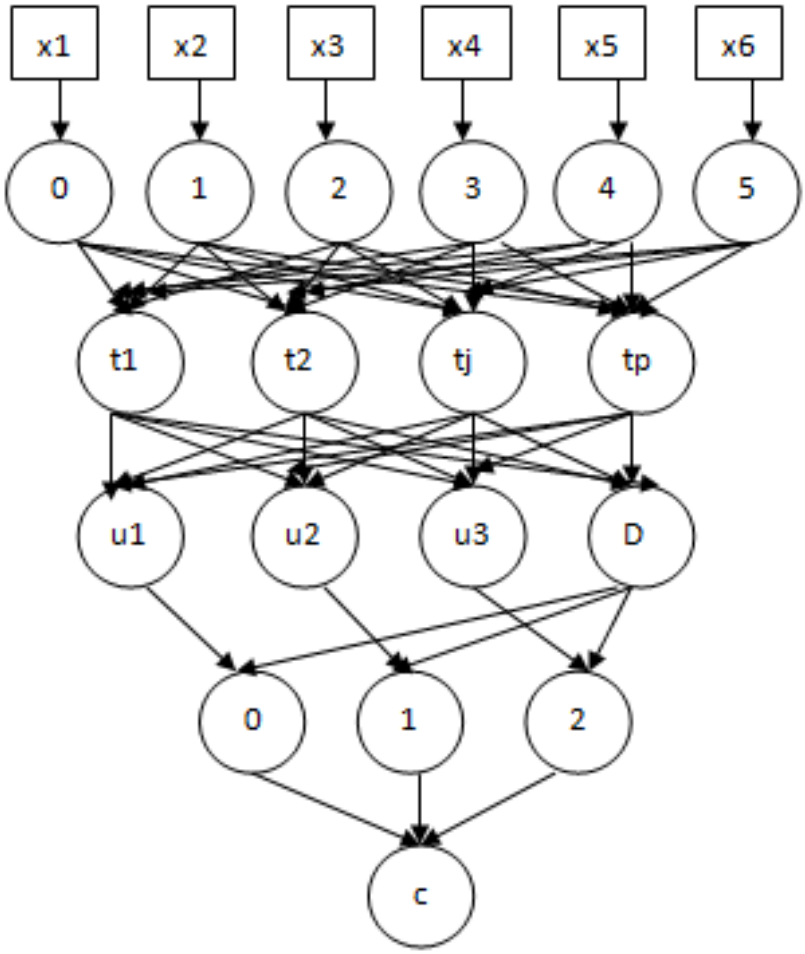
Gambar 3.5 Diagram FSM L-GCNN pada NPC

Dalam penelitian ini, NPC dapat melakukan 3 aksi yakni : *single attack*, *keep distance* dan *single attack in group*ing.

- *Single attack*ing : ketika agen berada pada posisi dan keadaan yang bagus untuk menyerang pemain
- *Keep distance* : agen menjauh dari pemain serta mencoba menghindari sumber pemain
- *Single attack in group*ing : ketika aksi "*single attack in group*ing" agen mencoba untuk tetap dekat ke agen lain didaerah sekitar mereka lalu menyerang secara berkelompok.

3.3. Desain penentuan perilaku NPC dengan L-GCNN

Dalam metode L-GCNN untuk penelitian ini, dibutuhkan beberapa *input* yang nantinya akan digunakan untuk proses pembelajaran serta *output* perilaku yang memungkinkan NPC mengalahkan pemain. Gambar 3.6 merupakan desain L-GCNN yang terdiri dari 5 *layer* yakni *input*, pattern, summation, normalization dan *output* seperti gambar 2.1 dengan menggunakan persamaan 2.3 – 2.16,dimana terdapat 6 *input* serta 3 *output* dengan 1 node diakhir *layer output* seperti gambar 3.4.



Gambar 3.6 Struktur L-GCNN untuk NPC

3.3.1. Desain *Input* dan *Output*

Input dalam penelitian terdiri dari 6 *input* :

- *Input* 0 (x1) : kesehatan NPC
- *Input* 1 (x2): jarak dengan pemain
- *Input* 2 (x3): pemain terlibat dengan NPC lain atau tidak (enganged)
- *Input* 3 (x4): daya serang NPC
- *Input* 4 (x5): jumlah teman NPC yang dekat
- *Input* 5 (x6): level NPC

Output dalam penilitian ini terdiri dari 3 *output* :

- *Output* 0 : *single attack*/ menyerang

- *Output* 1 : *Single attack in group*/ menyerang bersama (lebih dari 1 NPC)
- *Output* 2 : *keep distance*/ menjauh (bertahan)

3.3.2. Basic *Training*

Langkah pertama dalam menginisialisai jaringan saraf pada *game* adalah menemukan serangkaian pasangan state/action yang baik untuk offline *training*. set pelatihan ini nantinya dapat digunakan untuk mengatur nilai yang membantu agen memulai permainan. Menemukan data *training* yang baik sangat dibutuhkan, oleh karena itu tidak ada algoritma yang bagus untuk menetukan pasangan state/action. Untuk *output*, digunakan nilai 0,9 dan 0,1 agar terlihat dengan jelas node yang diaktifkan dan tidak diaktifkan. Tabel 3.1 menunjukkan contoh data *training* untuk penelitian ini.

Data set terdiri dari 6 parameter dari NPC yang berbeda dimana semua data diskalakan menjadi nilai antara 0 sampai 1 agar tidak terjadi perbedaan nilai yang signifikan antar parameter. Parameter pertama yakni health (kesehatan) NPC dengan rentang nilai 0-100. Parameter yang kedua yakni jarak antara NPC dengan pemain dengan rentang nilai 0-30. Parameter ketiga yakni enganged maksudnya apakah pemain sedang terlibat aksi dengan NPC yang lain atau tidak dengan nilai 0 ketika tidak terlibat dan 1 ketika terlibat. Parameter yang keempat yakni daya serang NPC dengan rentang nilai 0-25 selanjutnya jumlah teman NPC yang terdekat dengan jumlah NPC sebanyak 4 dan parameter yang terkahir yakni level NPC dengan rentang nilai 0-10

Tabel 3.1 Basic data *training*

State/*input*						Action / *output*		
0	1	2	3	4	5	0	1	2
0.5	0.2	0	0.3	0	0	0.1	0.1	0.9
0.5	0.2	1	0.3	2	1	0.1	0.9	0.1
0.2	0.2	0	0.5	1	3	0.9	0.1	0.1
0.2	0.2	1	0.5	0	1	0.1	0.1	0.9
0.8	0.8	0	0.1	1	2	0.9	0.1	0.1

Langkah-langkah membuat pasangan state/action untuk data *training* Tabel 3.1 adalah sebagai berikut :

1. Bangkitkan nilai acak untuk data *training*
2. Skalakan nilai menjadi rentang nilai 0 sampai 1
3. Menerjemahkan state ke nilai *input*
4. Memilih aksi yang diinginkan untuk state
5. Menerjemahkan aksi ke nilai *output*
6. Petakan state ke aksi
7. Ulangi lagi sampai mencapai jumlah data *training* yang diinginkan

3.3.3. Algoritma L-GCNN

L-GCNN memiliki dua proses yaitu proses pelatihan dan proses uji coba. Pada proses pelatihan *input*-nya adalah epoch (maksimal iterasi), learning rate (lr), data *training* dan error minimum (α_{te}), sedangkan *output*nya adalah nilai *parameter smoothing* (σ). pada proses uji coba *input*-nya adalah *parameter smoothing* yang didapatkan dari proses pelatihan, sedangkan *output*nya dari proses uji coba adalah kelas.

Algoritma 1. L-GCNN untuk proses pelatihan

Input : epoch, lr, data *training*, α_{te}

Output : parameter smoothing

Inisialisasi parameter smoothing σ dan ymax

While iteration ≤ epoch

For setiap data uji coba t_j

if iterasi > 1

if $\left(\sigma_j + lr * \frac{\partial e}{\partial \sigma_{id}}\right) > 0$

update σ_j dengan $\frac{\partial e}{\partial \sigma}$

temukan jarak euclidian antara data *input* dan data *training*, $d\ (j, i)$

lakukan fungsi aktifasi RBF, *r (j)*

for setiap kelas; i

hitung diverge effect term, $d\ (j,i) = \ e^{(y(j,i)-ymax)} * y(j,i)$

hitung $u_i \ = \ \sum_{j=1}^{p} d(j,i) * r(j)$ and $D = \ \sum_{j=1}^{p} r(j)$

hitung nilai *layer* neuron normalisasi ; $c_i = \ \frac{u_i}{D}$

end-for

temukan pemenang neuron dan nilainya; $[o, id] = \max\ (c)$

untuk update diverge effect term nilai neuron pemenang yang tersimpan; $cmax$ (iteration) = c_{id}

hitung logarithmic cost;

$e = \ (y(z,id) * \log(c_{id})) + \ (1 - y(z,id)) * log(1 - \ c_{id})$

dimana z menunjukkan *input* ke- z,

end-for

$y_{max} = \max(cmax)$

increment iterasi

if $te \ \leq \ \alpha_{te}$

stop *training*

end-while

Algoritma 2. L-GCNN untuk proses uji coba

Input : parameter smooting , data testing

Output : kelas

for setiap data *training*; t_j

temukan jarak euclidian antara data tes dan data *training*, $d\ (j,i)$

tentukan dengan fungsi aktifasi RBF, $r(j)$

for setiap kelas; i

hitung diverge effect term, $d\ (j,i) = \ e^{(y(j,i)-ymax)} * y(j,i)$

hitung $u_i \ = \ \sum_{j=1}^{p} d(j,i) * r(j)$ dan $D = \ \sum_{j=1}^{p} r(j)$

hitung nilai neuron *layer* normalisasi; $c_i = \ \frac{u_i}{D}$

end-for

temukan neuron pemenang dan nilainya; $[o, id] = \max\ (c)$

3.4. Skenario Pengujian

Pengujian dilakukan dengan membandingkan hasil penelitian menggunakan metode yang diuji dengan metode *Multi layer Perceptron*, *Radial Basis Function Neural Network* dan *Support Vector Machine.* Parameter yang diuji adalah tingkat akurasi. Untuk metode L-GCNN dan NN diuji cobakan menggunakan bahasa C# dalam Unity dalam menentukan waktu pelatihan sedangkan yang lain yakni RBNN, SVM dan NN untuk akurasi diuji cobakan menggunakan Weka dengan parameter standar yang ada di Weka.

Pengujian dilakukan dengan 5 skenario yakni :

1. Menggunakan Ten-fold cross validation
2. Membagi data set menjadi 2 bagian yakni 50% data *training* dan 50% data testing.
3. Membagi data set menjadi 2 bagian yakni 60% data *training* dan 40% data testing
4. Membagi data set menjadi 2 bagian yakni 70% data *training* dan 30% data testing
5. Membagi data set menjadi 2 bagian yakni 80% data *training* dan 20% data testing

3.5. Apparatus Pengujian

Pengujian yang dilakukan pada penelitian ini di jalankan pada hardware dengan spesifikasi sebagai berikut:

- Laptop HP Pavilion 14
- *Processor Core* i5-4210U 1.7 ~ 2.4 GHz
- RAM : 4096MB
- Windows 8.1 Pro 64-bit
- VGA : NVDIA GeForce 840M 3964MB
- Unity 2014.1.17

(*Halaman ini sengaja dikosongkan*)

BAB IV
HASIL PENGUJIAN

Penelitian ini diuji cobakan pada *game* berjenis RPG yang dikembangkan menggunakan aplikasi Unity dengan *environment* seperti gambar 4.1.

Gambar 4.1 *Game* RPG

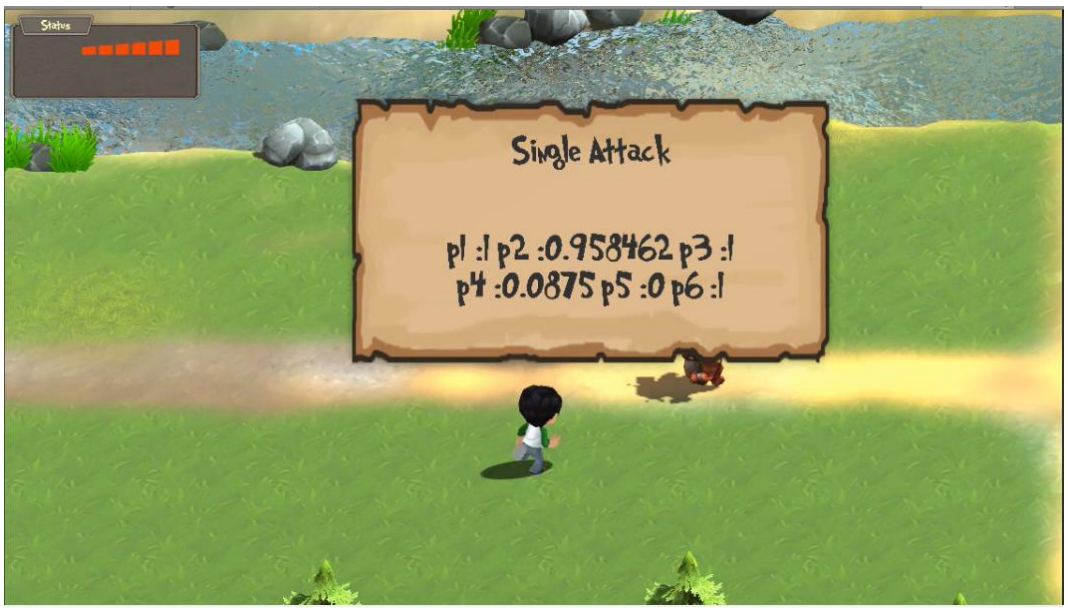
Data set yang digunakan sebanyak 400 data yang didapat dari pemilihan aksi sesuai dengan parameter NPC dengan langkah-langkah yang ada di bab 3. Tabel 4.1 merupakan 10 sampel data set.

Tabel 4.1 Sampel data set

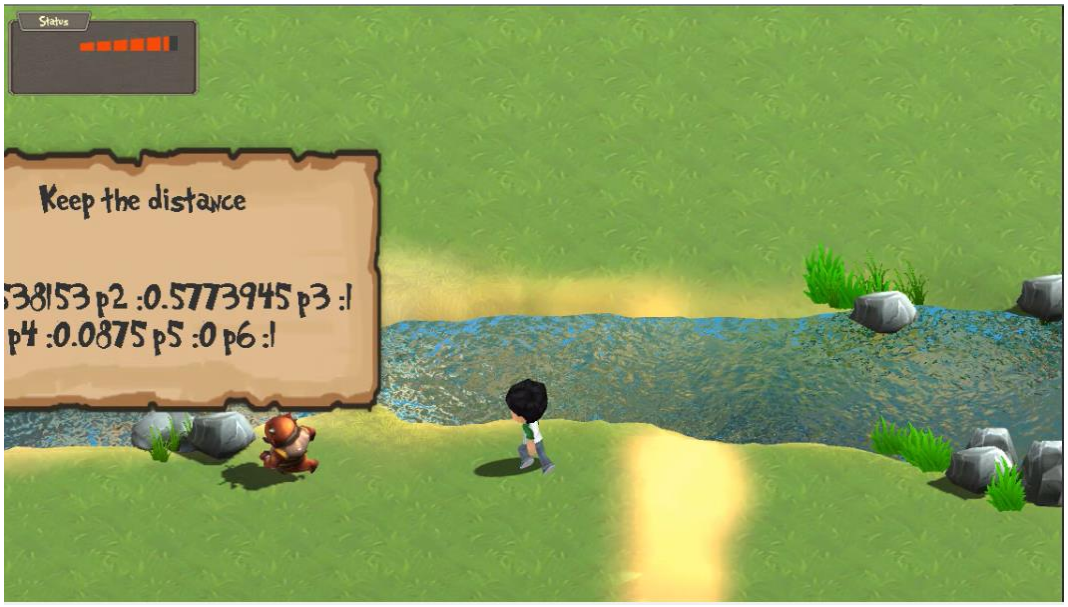
Data	Health	Jarak	Agen terlibat?	Daya serang	Jumlah teman	Level	Aksi
	P1	P2	P3	P4	P5	P6	Y
1	0.81	0.4	0	0	0.25	0.6	0
2	0	0.533333	1	0.04	0.75	0.2	1
3	0.15	0.5	1	0.32	0	0.9	2
4	0.47	0.633333	0	0.96	1	0.4	1
5	0.26	0.666667	1	0.76	0	0.5	2
6	0.77	0.066667	0	0.44	0	0.1	0
7	0.1	0.633333	0	0.4	0.75	0.4	1
8	0.05	0.233333	1	0.88	0.75	1	1
9	0.26	0.166667	0	1	0.75	0.8	1
10	0.31	0.333333	1	0.16	0	0.9	2

Data set terdiri dari 6 parameter dari NPC yang berbeda dimana semua data diskalakan menjadi nilai antara 0 sampai 1 agar tidak terjadi perbedaan nilai yang signifikan antar parameter. Parameter pertama yakni health (kesehatan) NPC dengan rentang nilai 0-100. Parameter yang kedua yakni jarak antara NPC dengan pemain dengan rentang nilai 0-30. Parameter ketiga yakni enganged maksudnya apakah pemain sedang terlibat aksi dengan NPC yang lain atau tidak dengan nilai 0 ketika tidak terlibat dan 1 ketika terlibat. Parameter yang keempat yakni daya serang NPC dengan rentang nilai 0-25 selanjutnya jumlah teman NPC yang terdekat dengan jumlah NPC sebanyak 4 dan parameter yang terakhir yakni level NPC dengan rentang nilai 0-10.

Hal pertama yang dilakukan adalah menguji metode *Neural Network* dengan struktur *multilayer* perceptron menggunakan algoritma *backpropagation* yang selanjutnya dibandingkan dengan metode yang diusulkan yakni dengan melihat waktu training dan tingkat akurasi. Gambar 4.2 merupakan aksi *single attack* yang dihasilkan ketika P1=1, P2=0.9, P3=1, P4=0.08, P5=0 dan P6=1. Kemudian gambar 4.3 merupakan aksi *keep the distance* yang dihasilkan ketika P1=0.53, P2=0.57, P3=1, P4=0.08, P5=0 dan P6=1. Sedangkan gambar 4.3 merupakan aksi *attack in group* yang dihasilkan ketika P1=0.53, P2=0.57, P3=1, P4=0.08, P5=0 dan P6=1.



Gambar 4.2 aksi *single attack*



Gambar 4.3 aksi *keep the distance*

Gambar 4.4 aksi *attack in group*

4.1 Hasil Pengujian NN

Metode *Neural Network* yang diuji terdiri dari 6 *layer input*, 3 *hidden layer* dan 1 *output layer* dengan nilai alpha 0.9 dan menggunakan fungsi sigmoid sebagai fungsi aktifasi. Parameter yang diujikan adalah tingkat akurasi dan waktu *training*, didapatkan hasil pengujian akurasi 84% dan waktu *training* 2.3 detik

seperti tabel 4.2 dengan 10-fold cross validation. Metode NN diuji cobakan menggunakan Weka dengan ketentuan standar yang ada di Weka.

Tabel 4.2 Ten-fold cross validation

Akurasi	Waktu *training*
84 %	2.36

4.2 Hasil Pengujian L-GCNN

Dalam metode L-GCNN terdapat dua proses yakni proses *training* dan proses *testing*. Pada proses *training*, *input* yang dibutuhkan yakni data *training*, epoch 200, learning rate 0.3 dan error minimum 0.1. Sedangkan *output* dari proses *training* adalah smoothing parameter yang nantinya digunakan dalam proses testing. Pada proses testing, *input* yang dibutuhkan adalah data testing dan nilai parameter smoothing optimal hasil dari proses *training* dan *output*nya berupa kelas. Struktur yang digunakan dalam metode LGCNN terdiri dari 6 neuron di *layer input*, 200 neuron di *layer* pattern, 4 neuron di *layer* summation, 3 *layer* normalization dan 1 neuron di *layer output*.

Langkah pertama yakni mengirimkan data *training* dari *layer input* ke *layer* pattern dimana satu neuron mewakili 1 data *training*. Lalu pada *layer* pattern dilakukan perhitungan jarak euclidian ($dist\ (j)$) menggunakan persamaan 2.3 setelah itu menghitung fungsi aktifasi RBF ($r(j)$) sebagai *output* dari *layer* pattern menggunakan persamaan 2.4. Setelah mendapatkan nilai fungsi aktifasi RBF lalu menentukan data tersebut termasuk dalam kelas ke- i atau tidak ($y(j,i)$), kalau data *training* tersebut termasuk kelas ke - i maka diberi nilai 0.9 selainnya diberi nilai 0.1 menggunakan persamaan 2.5 diperoleh hasil tabel 4.3.

Tabel 4.3 Hasil perhitungan *layer* pattern

$dist\ (j)$	$r(j)$	$y(j,i)$		
		0	1	2
1.444118	0.723155	0.9	0.1	0.1
1.304032	0.746253	0.1	0.9	0.1
1.301593	0.746662	0.1	0.1	0.9
1.422572	0.72666	0.1	0.9	0.1

0.786645	0.838146	0.1	0.1	0.9
1.004263	0.798191	0.9	0.1	0.1
1.670263	0.687365	0.1	0.9	0.1
1.283333	0.749728	0.1	0.9	0.1
1.196889	0.764417	0.1	0.9	0.1
1.019853	0.795403	0.1	0.1	0.9

Setelah itu, masuk ke *layer* summation dimana terdiri dari 4 neuron yakni 3 neuron kelas dan 1 neuron denominator. Pada *layer* ini dilakukan perhitungan diverge effect term ($d\ (j,i)$) menggunakan persamaan 2.6 diperoleh hasil tabel 4.4.

Tabel 4.4 Hasil perhitungan *layer* summation

$d\ (j,i)$)			*u(i)*		
0	1	2	0	1	2
0.9	0.044933	0.044933	0.650839	0.032493	0.032493
0.044933	0.9	0.044933	0.033531	0.671628	0.033531
0.044933	0.044933	0.9	0.03355	0.03355	0.671996
0.044933	0.9	0.044933	0.032651	0.653994	0.032651
0.044933	0.044933	0.9	0.03766	0.03766	0.754332
0.9	0.044933	0.044933	0.718372	0.035865	0.035865
0.044933	0.9	0.044933	0.030885	0.618628	0.030885
0.044933	0.9	0.044933	0.033687	0.674756	0.033687
0.044933	0.9	0.044933	0.034347	0.687975	0.034347
0.044933	0.044933	0.9	0.03574	0.03574	0.715863

Setelah nilai $d\ (j,i)$ didapatkan, selanjutnya menghitung nilai numerator (u_i) dan denominator (D). Nilai numerator ($u_{(i}$) diperoleh dari jumlah kali antara $d\ (j,i)$ dan $r(j)$ seperti persamaan 2.7. hasil dari perhitungan numerator bisa dilihat di tabel 4.5 sedangkan nilai denominator (D) didapatkan dari jumlah keseluruhan dai nilai fungsi aktiasi RBF di *layer* pattern seperti persamaan 2.8, diperoleh nilai $D = 154.7583$.

Selanjutnya, masuk ke *layer* normalization. Pada *layer* ini tiap neuron membagi nilai numerator dengan denominator seperti persamaan 2.9 dimana c_i menunjukkan *output* dari *layer* normalization. didapatkan hasil seperti tabel 4.5.

Tabel 4.5 Hasil perhitungan *layer* normalization

kelas	Ci
0	0.217108
1	0.574381
2	0.198377

Layer yang terakhir adalah *layer output*. Nilai pemenang diperoleh dari nilai c_i yang paling besar. Dalam proses *training*, nilai max (c) digunakan untuk menghitung cost function e (persamaan 2.11) sedangkan dalam proses testing nilai max (c) menunjukkan aksi yang dilakukan NPC terhadap pemain. Kemudian pada tabel 4.6, jika nilai e masih belum mencapai batas minimum dan $ymax$ masih berada direntang 0.1- 0.9 maka iterasi tetap dilakukan dengan menghitung smothing parameter yang baru menggunakan persamaan 2.12 – 2.16.

Tabel 4.6 Hasil perhitungan update smoothing parameter

l(id)	*b(id)*	$\frac{\partial c_{id}}{\partial \sigma}$	$\frac{\partial e}{\partial \sigma}$	σ new
1164.969	651.6569	-0.11294219	0.606734	0.791347

Proses *training* menghasilkan nilai parameter smoothing yang paling optimal. Dari penelitian ini, didapatkan nilai parameter smoothing (σ) yang paling optimal adalah 4.34. Tabel 4.7 menunjukkan hasil pengujian metode LGCNN dengan menggunakan 10-fold cross validation.

Tabel 4.7 Ten-fold cross validation LGCNN

Akurasi	Waktu *training*
93%	6.356 detik

4.3 Perbandingan Metode L-GCNN

Pengujian dilakukan dengan 5 skenario yakni dengan 10-fold cross validation kemudian dengan membagi data set dengan prosentase seperti tabel 4.8 dimana untuk pengujian kedua dari data set dibagi menjadi 2 yakni 50% data

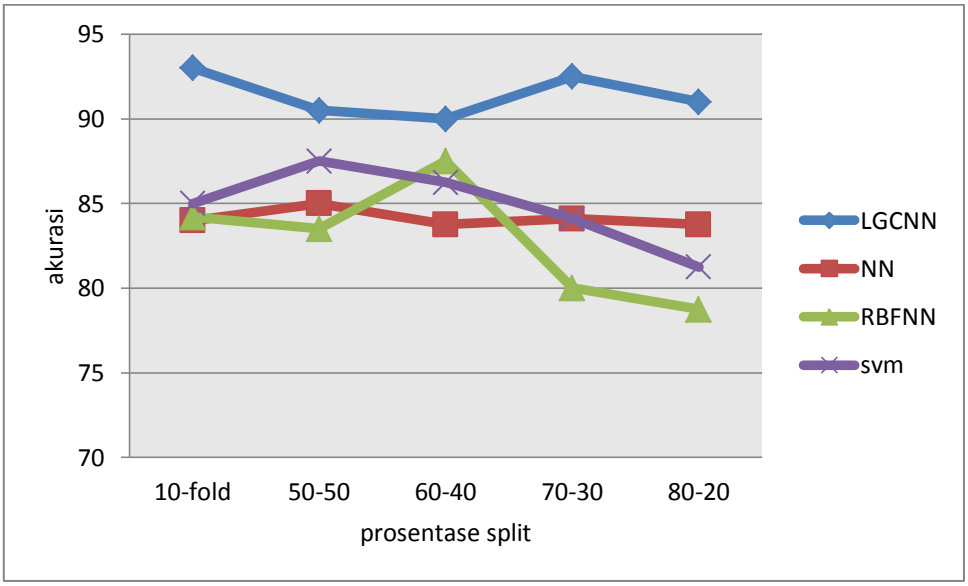
training dan 50% data testing, pengujian ketiga 60% data *training* dan 40% data testing, pengujian keempat 70% data *training* dan 30% data testing, pengujian kelima 80% data *training* dan 20% data testing.

Pengujian pertama dilakukan dengan membandingkan akurasi dengan 10-fold cross validation dan waktu *training* antara metode L-GCNN dan NN.Tabel 4.8 menunjukkan hasil pengujian dimana untuk tingkat akurasi, L-GCNN menunjukkan hasil yang lebih baik sebanyak 10% dari metode NN sedangkan untuk waktu *training* lebih cepat NN 33% dari pada L-GCNN dikarenakan pada L-GCNN mempunyai satu neuron untuk tiap data *training* pada *hidden layer* dimana NN memiliki lebih sedikit neuron pada *hidden layer*.

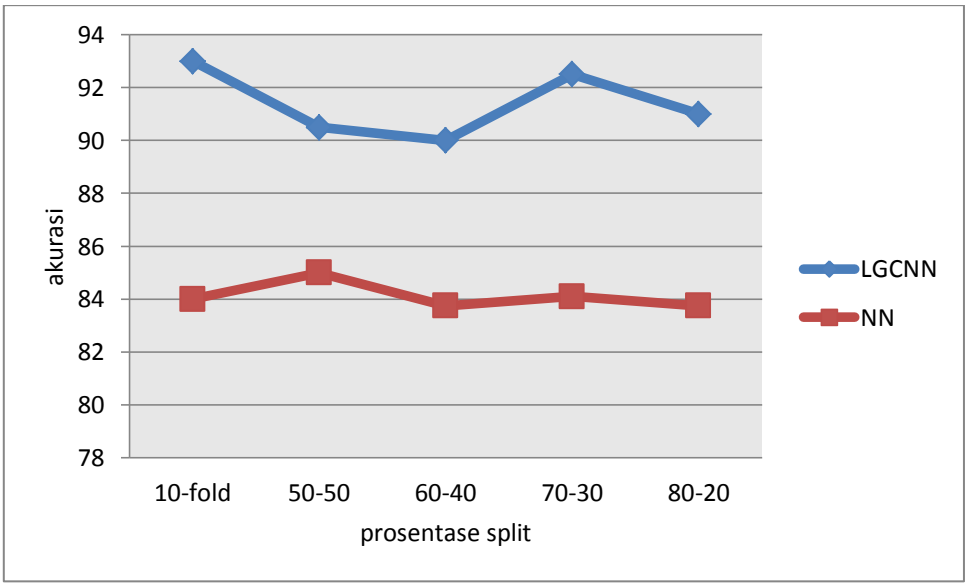
Tabel 4.8 Perbandingan akurasi dan waktu *training*

Metode	Akurasi	Waktu *training*
L-GCNN	93%	6.356 detik
NN	84%	2.36 detik

Pengujian selanjutnya yakni membandingkan metode L-GCNN dengan NN, RBFNN dan SVM dengan melihat tingkat akurasi pada setiap skenario pengujian. Gambar 4.1 menunjukkan hasil perbandingan L-GCNN dengan metode lain yang dibandingkan. L-GCNN memiliki performa yang lebih baik dari 3 metode lain yang dibandingkan. Terlihat pada gambar 4.1 L-GCNN dengan garis berwarna biru tingkat akurasi minimal 90% dan maksimal 93%, untuk metode NN minimal akurasi 83.75% dan maksimal 85%, untuk metode RBFNN minimal akurasi 80% dan maksimal 87% sedangkan untuk metode SVM minimal akurasi 81,25% dan maksimal 87,5%.



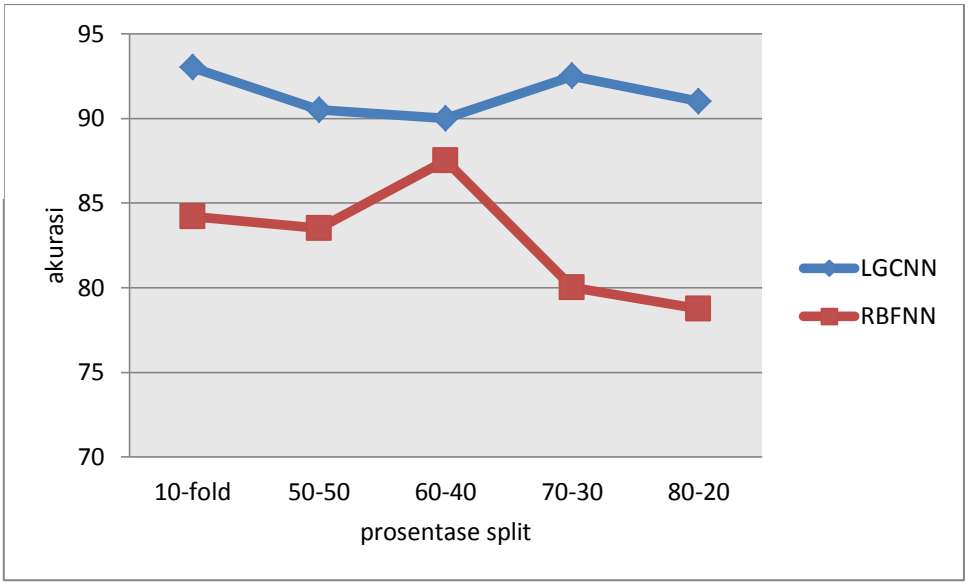
Gambar 4.5 Perbandingan L-GCNN



Gambar 4 6 Perbandingan L-GCNN dan NN

Gambar 4.2 menunjukkan hasil perbandingan akurasi dari metode L-GCNN dan NN dimana untuk 10-fold cross validation L-GCNN memiliki akurasi 93% dan NN 84%. Prosentase split 50% menghasilkan nilai akurasi 90,5% untuk L-GCNN dan 85% untuk NN. Prosentase split 60% menghasilkan nilai akurasi 90% untuk L-GCNN dan 83,75% untuk NN. Prosentase split 70% menghasilkan nilai akurasi 92.5% untuk L-GCNN dan 84.1% untuk NN sedangkan prosentase split 80% menghasilkan nilai akurasi 91% untuk L-GCNN dan 83.75% untuk NN.

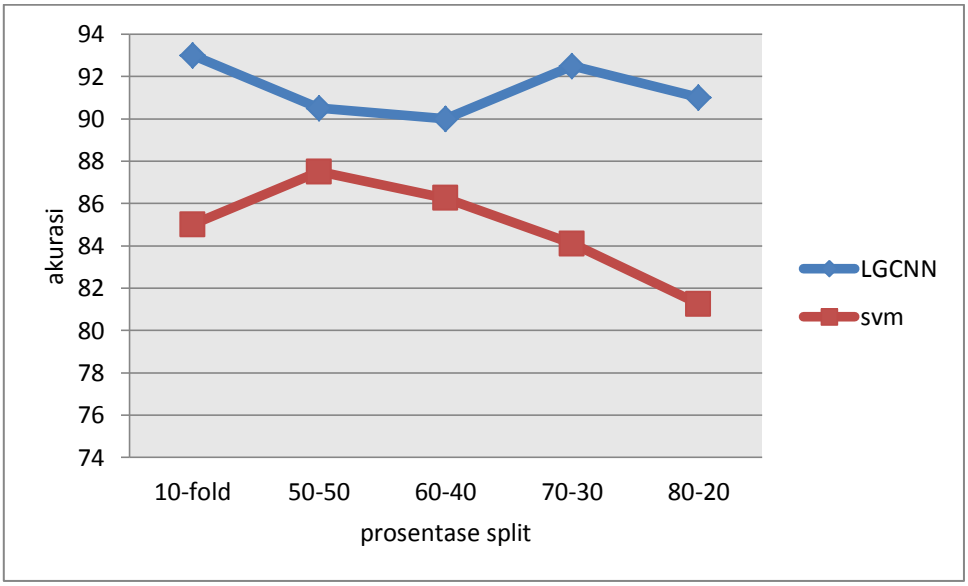
L-GCNN memilikin akurasi yang lebih baik dari NN karena dalam proses L-GCNN terdapat proses enkapsulasi data pada *layer* summation. Jika data ke-*j* memiliki kelas yang sama maka akan dikelompokkan menurut kelasnya.



Gambar 4.7 Perbandingan L-GCNN dan RBFNN

Gambar 4.3 menunjukkan hasil perbandingan akurasi dari metode L-GCNN dan RRBFNN dimana untuk 10-fold cross validation L-GCNN memiliki akurasi 93% dan RBFNN 84.2%. Prosentase split 50% menghasilkan nilai akurasi 90,5% untuk L-GCNN dan 83.5% untuk RBFNN. Prosentase split 60% menghasilkan nilai akurasi 90% untuk L-GCNN dan 83,75% untuk RBFNN. Prosentase split 70% menghasilkan nilai akurasi 92.5% untuk L-GCNN dan 80% untuk RBFNN sedangkan prosentase split 80% menghasilkan nilai akurasi 91% untuk L-GCNN dan 78.75% untuk RBFNN.

L-GCNN merupakan pengembangan dari RBFNN sehingga untuk kompleksitas waktu *training* tidak jauh beda dimana L-GCNN memiliki neuron sejumlah data sedangkan RBFNN menambahkan neuron di *hidden layer* jika dibutuhkan. Namun dalam segi akurasi, L-GCNN memiliki performa yang lebih baik dari RBFNN karena proses enkapsulasi data yang dilakukan di *layer* summation.



Gambar 4.8 Perbandingan L-GCNN dan SVM

Gambar 4.3 menunjukkan hasil perbandingan akurasi dari metode L-GCNN dan SVM dimana untuk 10-fold cross validation L-GCNN memiliki akurasi 93% dan SVM 85%. Prosentase split 50% menghasilkan nilai akurasi 90,5% untuk L-GCNN dan 87.5% untuk SVM. Prosentase split 60% menghasilkan nilai akurasi 90% untuk L-GCNN dan 86.25% untuk SVM. Prosentase split 70% menghasilkan nilai akurasi 92.5% untuk L-GCNN dan 84.1% untuk SVM sedangkan prosentase split 80% menghasilkan nilai akurasi 91% untuk L-GCNN dan 81.25% untuk SVM.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Setelah dilakukan pengujian sesuai dengan scenario, didapatkan hasil bahwa metode L-GCNN yang diuji memiliki akurasi yang lebih baik dari 3 metode yang dibandingkan yakni dengan tingkat akurasi rata-rata 7% lebih baik dari NN dan SVM serta 8% lebih baik dari RBFNN. Dari data tersebut dapat di tarik kesimpulan juga bahwa penggunaan parameter smoothing pada L-GCNN yang menunjukkan radius dengan tetangga yang paling efektif dapat meningkatkan akurasi dari metode. Jika data tersebut termasuk dalam kelas yang sama maka dapat dikelompokkan menurut kelasnya.

5.2. Saran

Dari hasil uji coba yang dilakukan,, terkadang data uji coba mengalami error berupa nilai cost function yang infinity maupun berupa NaN, hal ini dikarenakan pemilihan inisial smoothing parameter yang salah dan learning rate yang tidak tepat. Sehingga peneliti memberikan saran pada peneliti selanjutnya untuk mengoptimalisasikan nilai initial smoothing parameter dan learning rate.

Metode NN memiliki waktu training yang lebih cepat namun akurasi yang kurang baik oleh karena itu metode ini masih memungkinkan digunakan dengan menggabungkan dengan metode lain

(*Halaman ini sengaja dikosongkan)*

DAFTAR PUSTAKA

Edmond S. L. Ho & Toku K. (2011). A Finite State Machine based on Topology Coordinates for Wrestling *Game*s. Computer Animation and Virtual Worlds.

Rosalina E.D, Alfian M, Azhari M, dkk. (2015). First Aid Simulation *Game* with Finite State Machine Model. *International Conferences on Information, Communication Technology and System* (ICTS). IEEE.

Tito Bimantoro & Hanny Haryanto. (2016). Pemodelan Perilaku Musuh Menggunakan Finite State Machine pada *Game* Pengenalan Unsur Kimia. *Journal of Applied Intelligent System*, Vol.1, No. 3, 210-219.

Yunifa M.A., Fressy N, dkk. (2011). Desain Perubahan Perilaku pada NPC *Game* Menggunakan Logika Fuzzy. Seminar nasional *Electrical Informatics and ITS Education*.

Supeno Mardi S. N, Ika Widiastuti, M. Hariadi & M. H. (2013). Fuzzy Coordinator based Intelligent Agents for Team Coordination in Close Combat *Game*. *Theoritical and Applied Information Technology*, vol. 51.

Fahrul P.P., Ahmad Zainul F & M. Hariadi. (2014). Perilaku otonom dan Adaptif Non Pemain Character Musuh pada *Game* 3 Dimensi Menggunakan Fuzzy State Machine dan Rule Based System. Seminar Nasional Teknologi Informasi & Komunikasi Terapan (SEMANTIK).

Andreas Pfeifer. (2009). Creating Adaptive *Game* AI in a Real Time Continuous Environment using *Neural Network*s. *Thesis on Technische Universitat Darmstadt*.

Diwas sharma & Udit Kr.Chakraborty. (2013). Artificial *Neural Network* Based Adaptive Chess Playing Machine. *International Journal of Advanced Research in Computer Science* (IJARCS), Vol. 4, No. 4.

Buse Melis & Mutlu Avci. (2013). Generalized Classifier *Neural Network*. ELSEVIER, 18-26.

Buse Melis & Mutlu Avci. (2014). Logarithmic Learning for Generalized Classifier *Neural Network*. ELSEVIER, 133-10..

Umarov L. & Mozgovoy M. (2012). Belieavable and Effective AI Agents in Virtual Worlds : Current state and future perspective. *Interantional Journal of Gaming and Cumputer-Mediated Simulations*, Vol. 4, 37-59.

Ian Millington & john Funge. (2009). Arificial Intelligence for *Game*s (Second Edition). *Morgan Kaufmann*.

Bryant, B. D. and Miikkulainen, R. (2003). Neuroevolution for Adaptife teams. *Proceedings of the 2003 Congress on Evolutionary Computation*, Vol.3, 2194-2201. IEEE.

Fogel, D. B., Hays, T.J. & Johnson. (2004). A platform for Evolving Characters in Competitive *Game*s. *Proceedings of the 2004 Congress on Evolutionary Computation*, 1420-126. IEEE.

Hermawan, A. (2006). *Jaringan Saraf Tiruan Teori dan Aplikasi*. Yogyakarta : Penerbit Andi.

Puspaningrum, D. (2006). *Pengantar Jaringan Saraf Tiruan*. Yogyakarta : Penerbit Andi